AA-DC94A-TE

OFFICE MENU

Application Programmer's Reference, Volume 3: Applications

digital™

AA-DC94A-TE
First Printing, February 1985

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1985 by Digital Equipment Corporation. All rights reserved.

The postage prepaid Reader's Comments Form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

digital™	DECSYSTEM-20	RSTS
ALL-IN-1	DECtalk	RSX
DATATRIEVE	DECUS	UNIBUS
DEC	DECwriter	VAX
DECmail	DIBOL	VMS
DECmate	MASSBUS	VT
DECnet	PDP	Work Processor
DECspell	P/OS	WPS-PLUS
DECsystem-10	Professional	
	Rainbow	

Contents

Volume 1: Flow Control

Preface

1 Overview

1.1 Introduction 1-1
1.2 ALL-IN-1 in the VAX Information Architecture 1-3
1.3 ALL-IN-1 Architecture and Basic Concepts 1-3
1.3.1 Form and Function 1-5
1.3.2 How ALL-IN-1 Is Organized 1-6

2 ALL-IN-1 Initialization and Flow Control

2.1 Introduction 2-1
2.2 ALL-IN-1 Initialization 2-2
2.2.1 Partial Initialization 2-3
2.3 ALL-IN-1 Office Menu Flow Control 2-3

3 Form Processing

3.1 Introduction 3-1
3.2 Form Libraries 3-2
3.2.1 Searching Form Libraries 3-3

3.3 The FORM and SHOW_FORM Functions 3-3
3.3.1 The FORM Function 3-3
3.3.2 The SHOW_FORM Function 3-4
3.4 ALL-IN-1 Named Data Conventions 3-4
3.4.1 ALL-IN-1 Form .TYPEs 3-5
3.4.2 The .MORE Directive 3-7
3.4.3 Searching for Named Data 3-7
3.4.4 Form Qualifiers 3-8
3.4.5 Common, or Shared, Form Qualifiers 3-10
3.5 The MENU Form 3-24
3.5.1 Menu Named Data 3-25
3.5.2 Menu Form Qualifiers 3-25
3.5.3 Processing Menus 3-28
3.5.4 Stacking Menu Options 3-32
3.5.5 Invoking Menu Options Indirectly 3-33
3.5.6 Creating and Processing Complex Menus 3-33
3.5.7 Menu Stacks and Menu Levels 3-36
3.6 The Entry Form 3-37
3.6.1 Entry Form Named Data 3-37
3.6.2 The Data Entry Process: Creating Files 3-38
3.6.3 The Data Entry Process: Accessing Data Files 3-44
3.6.4 Entry Form Qualifiers 3-44
3.6.5 Processing Entry Forms 3-54.1
3.7 The Argument Form 3-56
3.7.1 Argument Form Named Data 3-56
3.7.2 Argument Form Qualifiers 3-57
3.8 The Calc and Desk Forms 3-57
3.8.1 Calc Form Named Data 3-57
3.8.2 Calc Form Qualifiers 3-58
3.8.3 The Desk Calculator Form 3-58
3.9 The Search Form 3-60
3.9.1 Search Form Named Data 3-61
3.9.2 Search Expressions in the .TYPE Directive 3-63
3.9.3 Search Form Qualifiers 3-65
3.9.4 Searching for Strings: The .SFILE Directive 3-65
3.9.5 Formatting Record Collections: The .SOUT Directive 3-67
3.9.6 Search Form Processing 3-68
3.10 The Select Form 3-68
3.10.1 Select Form Named Data 3-69
3.10.2 Select Form Qualifiers 3-69
3.10.3 Selecting and Processing Records 3-71
3.11 EXIT SCREEN and the PRIOR Form 3-75

3.12 Dynamically Redefining Named Data 3-75
3.12.1 The Form .TYPE Parameter in FORM Function Calls 3-76
3.12.2 The /RESET Form Qualifier 3-77
3.12.3 Dynamically Redefining Entry Forms 3-78
3.13 Multipage Form Sets 3-78
3.14 Nonstandard Forms 3-80

4 Field Processing

4.1 Introduction 4-1
4.2 The FIELD Function 4-2
4.3 Defining Fields 4-2
4.4 Types of Field Qualifiers 4-4
4.5 /AUTO 4-5
4.5.1 Examples 4-5
4.6 /BLANK 4-5
4.6.1 Example 4-6
4.7 /CLEAR 4-6
4.7.1 Examples 4-6
4.8 /COMPUTE 4-7
4.8.1 Examples 4-7
4.9 /COPY 4-8
4.9.1 Examples 4-9
4.10 /DECIMAL 4-10
4.10.1 Examples 4-11
4.11 /DOC 4-11
4.11.1 Examples 4-12
4.12 /FILE_CHECK 4-14
4.12.1 Examples 4-15
4.13 /GET_SAVE 4-16
4.13.1 Examples 4-17
4.14 /HARD 4-18
4.14.1 Examples 4-18
4.15 /INVALID 4-19
4.15.1 Examples 4-22
4.16 /LIST 4-22
4.16.1 Examples 4-23
4.17 /OPTIONAL 4-23
4.17.1 Examples 4-24
4.18 /POST_FUNCTION 4-24
4.18.1 Examples 4-25
4.19 /PRE_FUNCTION 4-25
4.19.1 Examples 4-26

4.20 /PROMPT 4-26
4.20.1 Example 4-27
4.21 /PUT_SAVE 4-27
4.21.1 Examples 4-27
4.22 /RECOGNITION 4-28
4.22.1 Examples 4-30
4.23 /RSE_nnn 4-30
4.23.1 Examples 4-32
4.24 /SCROLL 4-33
4.24.1 /SCROLL with Indexed Files 4-35
4.24.2 SCROLL$FUNCTION 4-36
4.24.3 Example 4-40
4.24.4 Customized Scrolling 4-43
4.25 /SHOW 4-45
4.25.1 Examples 4-46
4.26 /STORE 4-46
4.27 /UNIQUE 4-47
4.27.1 Example 4-47
4.28 /USE_FORM 4-47
4.29 /VALID 4-47
4.29.1 Examples 4-50
4.30 Data Validation 4-51
4.30.1 Combining Validation Qualifiers 4-54
4.30.2 Controlling Record Collection Output 4-55
4.31 Default Keyboard Definitions 4-56
4.32 Defining the Keyboard 4-63
4.33 Terminator Processing 4-66
4.34 Field Processing 4-69

5 Generic Menu Design

5.1 The Primary Menu Form 5-1
5.1.1 Current Items 5-3
5.1.2 Select (SEL) 5-3
5.1.3 Create (C) 5-4
5.1.4 Edit (E) 5-5
5.1.5 Delete (D) 5-5
5.1.6 Print (P) 5-6
5.1.7 Read (R) 5-6
5.1.8 Index (I) 5-6
5.1.9 Option x (Ox) 5-6
5.1.10 Training (TR) 5-6
5.2 More Menu Options (M) 5-7
5.3 Secondary Menu Form - Additional Menu Options 5-7

Volume 2: Functions

Preface

6 Functions

6.1 Introduction 6-1
6.2 Function Call Syntax 6-2
6.3 Stacking Functions 6-4
6.3.1 Trapping Responses and Error Conditions 6-6
6.3.2 Function Loops 6-7
6.3.3 Forcing Displays 6-7
6.3.4 Other Features 6-7
6.4 Pipelining Functions 6-9
6.5 User-Defined Functions 6-10
6.6 Symbol Substitution in Functions 6-12
6.7 ACMSTASK 6-13
6.7.1 Syntax 6-13
6.7.2 Description 6-13
6.7.3 Function Parameters 6-13
6.7.4 Function Qualifiers 6-14
6.7.5 Examples 6-14
6.7.6 Special Features and Restrictions 6-15
6.8 ADE 6-16
6.8.1 Syntax 6-16
6.8.2 Description 6-16
6.8.3 Function Parameters 6-16
6.9 AI_CHECK 6-17
6.9.1 Syntax 6-17
6.9.2 Description 6-17
6.9.3 Function Parameters 6-17
6.9.4 Function Qualifiers 6-17
6.9.5 Examples 6-18
6.9.6 Special Features and Restrictions 6-18
6.10 APPEND 6-19
6.10.1 Syntax 6-19
6.10.2 Description 6-19
6.10.3 Function Parameters 6-19
6.10.4 Examples 6-20
6.11 ATTACH 6-21
6.11.1 Syntax 6-21

6.11.2 Description 6-21
6.11.3 Function Parameters 6-21
6.11.4 Function Qualifiers 6-21
6.11.5 Example 6-21
6.11.6 Special Features and Restrictions 6-22
6.12 BYE 6-23
6.12.1 Syntax 6-23
6.12.2 Description 6-23
6.12.3 Function Parameters 6-23
6.12.4 Function Qualifiers 6-23
6.12.5 Examples 6-24
6.12.6 Special Features and Restrictions 6-24
6.13 CABINET 6-25
6.13.1 Syntax 6-25
6.13.2 Description 6-25
6.13.3 Subfunction Descriptions 6-25
6.14 CALENDAR 6-26
6.14.1 Syntax 6-26
6.14.2 Description 6-26
6.14.3 Subfunction Descriptions 6-26
6.14A CHECK_DISKQUOTA 6-26.1
6.14A.1 Syntax 6-26.1
6.14A.2 Description 6-26.1
6.14A.3 Function Parameters 6-26.1
6.14A.4 Function Qualifiers 6-26.2
6.14A.5 Examples 6-26.2
6.14A.6 Special Features and Restrictions 6-26.2
6.15 CLEAR 6-27
6.15.1 Syntax 6-27
6.15.2 Description 6-27
6.15.3 Function Parameters 6-27
6.15.4 Function Qualifiers 6-27
6.15.5 Examples 6-28
6.15.6 Special Features and Restrictions 6-28
6.16 CLOSE_PRIOR 6-29
6.16.1 Syntax 6-29
6.16.2 Description 6-29
6.16.3 Function Parameters 6-29
6.16.4 Function Qualifiers 6-29
6.16.5 Examples 6-30
6.17 COMMAND 6-31
6.17.1 Syntax 6-31
6.17.2 Description 6-31
6.17.3 Function Parameters 6-31

6.17.4 Function Qualifiers 6-32
6.17.5 Examples 6-33
6.17.6 Special Features and Restrictions 6-34
6.18 COMPUTE 6-37
6.18.1 Syntax 6-37
6.18.2 Description 6-37
6.18.3 Function Parameters 6-37
6.18.4 Function Qualifiers 6-39
6.18.5 Examples 6-39
6.18.6 Special Features and Restrictions 6-41
6.19 CONTROL_C 6-42
6.19.1 Syntax 6-42
6.19.2 Description 6-42
6.19.3 Function Parameters 6-42
6.20 COPY 6-43
6.20.1 Syntax 6-43
6.20.2 Description 6-43
6.20.3 Function Parameters 6-43
6.20.4 Function Qualifiers 6-44
6.20.5 Examples 6-44
6.20.6 Special Features and Restrictions 6-44
6.21 CREATE 6-45
6.21.1 Syntax 6-45
6.21.2 Description 6-45
6.21.3 Function Parameters 6-45
6.21.4 Function Qualifiers 6-46
6.21.5 Examples 6-46
6.21.6 Special Features and Restrictions 6-46
6.22 CXP 6-47
6.22.1 Syntax 6-47
6.22.2 Description 6-47
6.22.3 Subfunction Descriptions 6-47
6.23 DATE_CONVERT 6-48
6.23.1 Syntax 6-48
6.23.2 Description 6-48
6.23.3 Function Parameters 6-48
6.23.4 Function Qualifiers 6-49
6.23.5 Examples 6-50
6.24 DCL 6-51
6.24.1 Syntax 6-51
6.24.2 Description 6-51
6.24.3 Function Parameters 6-51
6.24.4 Function Qualifiers 6-52
6.24.5 Examples 6-52
6.24.6 Special Features and Restrictions 6-52

6.25 DECIMAL 6-54
6.25.1 Syntax 6-54
6.25.2 Description 6-54
6.25.3 Function Parameters 6-54
6.25.4 Function Qualifiers 6-55
6.25.5 Examples 6-55
6.25.6 Special Features and Restrictions 6-55
6.26 DELETE_FILE 6-56
6.26.1 Syntax 6-56
6.26.2 Description 6-56
6.26.3 Function Parameters 6-56
6.26.4 Function Qualifiers 6-57
6.26.5 Examples 6-57
6.26.6 Special Features and Restrictions 6-57
6.27 DESTINATION 6-58
6.27.1 Syntax 6-58
6.27.2 Description 6-58
6.27.3 Function Parameters 6-58
6.27.4 Function Qualifiers 6-58
6.27.5 Special Features and Restrictions 6-58
6.28 DISPLAY 6-59
6.28.1 Syntax 6-59
6.28.2 Description 6-59
6.28.3 Function Parameters 6-59
6.28.4 Function Qualifiers 6-60
6.28.5 Examples 6-60
6.28.6 Special Features and Restrictions 6-60
6.29 DO 6-63
6.29.1 Syntax 6-63
6.29.2 Description 6-63
6.29.3 Function Parameters 6-63
6.29.4 Function Qualifiers 6-64
6.29.5 Examples 6-64
6.29.6 Special Features and Restrictions 6-64
6.30 DTR 6-65
6.30.1 Syntax 6-65
6.30.2 Description 6-65
6.30.3 Function Parameters 6-65
6.30.4 Function Qualifiers 6-66
6.30.5 Examples 6-66
6.30.6 Special Features and Restrictions 6-67

6.31 DUMP_CACHE 6-69
6.31.1 Syntax 6-69
6.31.2 Description 6-69
6.31.3 Function Parameters 6-69
6.31.4 Function Qualifiers 6-69
6.31.5 Examples 6-70
6.31.6 Special Features and Restrictions 6-70
6.32 EDIT 6-71
6.32.1 Syntax 6-71
6.32.2 Description 6-71
6.32.3 Function Parameters 6-71
6.32.4 Function Qualifiers 6-73
6.32.5 Special Features and Restrictions 6-73
6.33 EDT 6-74
6.33.1 Syntax 6-74
6.33.2 Description 6-74
6.33.3 Function Parameters 6-74
6.33.4 Function Qualifiers 6-74
6.33.5 Special Features and Restrictions 6-75
6.34 ERROR 6-76
6.34.1 Syntax 6-76
6.34.2 Description 6-76
6.34.3 Function Parameters 6-76
6.34.4 Function Qualifiers 6-76
6.34.5 Examples 6-77
6.34.6 Special Features and Restrictions 6-77
6.35 EXIT 6-78
6.35.1 Syntax 6-78
6.35.2 Description 6-78
6.35.3 Function Parameters 6-78
6.35.4 Function Qualifiers 6-78
6.35.5 Examples 6-79
6.35.6 Special Features and Restrictions 6-79
6.36 FHELP 6-80
6.36.1 Syntax 6-80
6.36.2 Description 6-80
6.36.3 Function Parameters 6-80
6.36.4 Function Qualifiers 6-80
6.36.5 Examples 6-80

6.37 FIELD 6-81
6.37.1 Syntax 6-81
6.37.2 Description 6-81
6.37.3 Function Parameters 6-81
6.37.4 Function Qualifiers 6-82
6.37.5 Examples 6-82
6.37.6 Special Features and Restrictions 6-82
6.38 FOR 6-83
6.38.1 Syntax 6-83
6.38.2 Description 6-83
6.38.3 Function Parameters 6-83
6.38.4 Boolean Expressions 6-85
6.38.5 Special DO Subfunctions 6-87
6.38.6 Examples 6-88
6.38.7 Special Features and Restrictions 6-90
6.39 FORCE 6-92
6.39.1 Syntax 6-92
6.39.2 Description 6-92
6.39.3 Function Parameters 6-92
6.39.4 Function Qualifiers 6-92
6.39.5 Examples 6-93
6.40 FORM 6-94
6.40.1 Syntax 6-94
6.40.2 Description 6-94
6.40.3 Function Parameters 6-94
6.40.4 Form Qualifiers 6-96
6.41 FORMAT 6-97
6.41.1 Syntax 6-97
6.41.2 Description 6-97
6.41.3 Function Parameters 6-97
6.41.4 Function Qualifiers 6-99
6.41.5 Examples 6-99
6.41.6 Special Features and Restrictions 6-100
6.42 GET 6-101
6.42.1 Syntax 6-101
6.42.2 Description 6-101
6.42.3 Function Parameters 6-102
6.42.4 Function Qualifiers 6-103
6.42.5 Examples 6-103
6.42.6 Special Features and Restrictions 6-107

6.43 HELP 6-108
6.43.1 Syntax 6-108
6.43.2 Description 6-108
6.43.3 Function Parameters 6-108
6.43.4 Function Qualifiers 6-109
6.43.5 Examples 6-110
6.43.6 Special Features and Restrictions 6-110
6.44 IFEXIT 6-112
6.44.1 Syntax 6-112
6.44.2 Description 6-112
6.44.3 Function Parameters 6-112
6.44.4 Function Qualifiers 6-112
6.44.5 Examples 6-113
6.44.6 Special Features and Restrictions 6-113
6.45 INCLUDE 6-114
6.45.1 Syntax 6-114
6.45.2 Description 6-114
6.45.3 Function Parameters 6-114
6.45.4 Function Qualifiers 6-114
6.45.5 Examples 6-115
6.45.6 Special Features and Restrictions 6-115
6.46 KEYPAD 6-116
6.46.1 Syntax 6-116
6.46.2 Description 6-116
6.46.3 Function Parameters 6-116
6.46.4 Function Qualifiers 6-116
6.46.5 Examples 6-117
6.46.6 Special Features and Restrictions 6-117
6.47 LIGHTS 6-118
6.47.1 Syntax 6-118
6.47.2 Description 6-118
6.47.3 Function Parameters 6-118
6.47.4 Examples 6-118
6.47.5 Special Features and Restrictions 6-119
6.48 LIST 6-120
6.48.1 Syntax 6-120
6.48.2 Description 6-120
6.48.3 Function Parameters 6-120
6.48.4 Function Qualifiers 6-121
6.48.5 Examples 6-121
6.48.6 Special Features and Restrictions 6-122

6.49 LOG 6-124
6.49.1 Syntax 6-124
6.49.2 Description 6-124
6.49.3 Function Parameters 6-124
6.49.4 Function Qualifiers 6-124
6.49.5 Examples 6-125
6.49.6 Special Features and Restrictions 6-125
6.50 LOGICAL 6-126
6.50.1 Syntax 6-126
6.50.2 Description 6-126
6.50.3 Function Parameters 6-126
6.50.4 Function Qualifiers 6-127
6.50.5 Examples 6-127
6.50.6 Special Features and Restrictions 6-128
6.51 MAIL 6-129
6.51.1 Syntax 6-129
6.51.2 Description 6-129
6.51.3 Subfunction Descriptions 6-129
6.52 MAKE_FILE_NAME 6-130
6.52.1 Syntax 6-130
6.52.2 Description 6-130
6.52.3 Function Parameters 6-130
6.52.4 Function Qualifiers 6-132
6.52.5 Examples 6-132
6.52.6 Special Features and Restrictions 6-133
6.53 MERGE 6-134
6.53.1 Syntax 6-134
6.53.2 Description 6-134
6.53.3 Function Parameters 6-134
6.53.4 Template Directives 6-137
6.53.5 Function Qualifiers 6-139
6.53.6 Examples 6-139
6.53.7 Special Features and Restrictions 6-140
6.54 MERGE_LINE 6-141
6.54.1 Syntax 6-141
6.54.2 Description 6-141
6.54.3 Function Parameters 6-141
6.54.4 Function Qualifiers 6-141
6.54.5 Examples 6-142
6.54.6 Special Features and Restrictions 6-142

6.55 MERGE_LIST 6-143
6.55.1 Syntax 6-143
6.55.2 Description 6-143
6.55.3 Function Parameters 6-143
6.55.4 Function Qualifiers 6-145
6.55.5 Examples 6-146
6.55.6 Special Features and Restrictions 6-146
6.56 MODE 6-147
6.56.1 Syntax 6-147
6.56.2 Description 6-147
6.56.3 Function Parameters 6-147
6.56.4 Function Qualifiers 6-148
6.56.5 Examples 6-148
6.56.6 Special Features and Restrictions 6-149
6.57 NEWDIR 6-150
6.57.1 Syntax 6-150
6.57.2 Description 6-150
6.57.3 Function Parameters 6-150
6.57.4 Function Qualifiers 6-151
6.57.5 Examples 6-151
6.57.6 Special Features and Restrictions 6-151
6.58 NEWLIB 6-152
6.58.1 Syntax 6-152
6.58.2 Description 6-152
6.58.3 Function Parameters 6-152
6.58.4 Function Qualifiers 6-153
6.58.5 Examples 6-153
6.58.6 Special Features and Restrictions 6-153
6.59 NEXT_LIST 6-155
6.59.1 Syntax 6-155
6.59.2 Description 6-155
6.59.3 Function Parameters 6-155
6.59.4 Function Qualifiers 6-155
6.59.5 Examples 6-156
6.59.6 Special Features and Restrictions 6-156
6.60 OA$CLI_GET_SYMBOL 6-157
6.60.1 Syntax 6-157
6.60.2 Description 6-157
6.60.3 Function Parameters 6-157
6.60.4 Function Qualifiers 6-157
6.60.5 Examples 6-158
6.60.6 Special Features and Restrictions 6-158

6.61 OA$CLI_GET_VALUE 6-159
6.61.1 Syntax 6-159
6.61.2 Description 6-159
6.61.3 Function Parameters 6-159
6.61.4 Function Qualifiers 6-159
6.61.5 Examples 6-160
6.61.6 Special Features and Restrictions 6-160
6.62 OA$CNV_CONVERT 6-161
6.62.1 Syntax 6-161
6.62.2 Description 6-161
6.62.3 Function Parameters 6-162
6.62.4 Function Qualifiers 6-162
6.62.5 Examples 6-162
6.62.6 Special Features and Restrictions 6-162
6.63 OA$CNV_DIST_LIST 6-163
6.63.1 Syntax 6-163
6.63.2 Description 6-163
6.63.3 Function Parameters 6-163
6.63.4 Examples 6-164
6.63.5 Special Features and Restrictions 6-164
6.64 OA$CNV_MERGE 6-165
6.64.1 Syntax 6-165
6.64.2 Description 6-165
6.64.3 Function Parameters 6-166
6.64.4 Function Qualifiers 6-166
6.64.5 Examples 6-166
6.64.6 Special Features and Restrictions 6-166
6.65 OA$CNV_FROM_SCROLL 6-167
6.65.1 Syntax 6-167
6.65.2 Description 6-167
6.65.3 Function Parameters 6-167
6.65.4 Function Qualifiers 6-167
6.66 OA$CNV_TO_SCROLL 6-168
6.66.1 Syntax 6-168
6.66.2 Description 6-168
6.66.3 Function Parameters 6-168
6.66.4 Function Qualifiers 6-168
6.67 OA$DSA_CACHE_STATUS 6-169
6.67.1 Syntax 6-169
6.67.2 Description 6-169
6.67.3 Function Parameters 6-169
6.67.4 Function Qualifiers 6-169
6.67.5 Examples 6-170
6.67.6 Special Features and Restrictions 6-170

6.68 OA$ENT_SET_xxx 6-171
6.68.1 Syntax 6-171
6.68.2 Description 6-171
6.68.3 Function Parameters 6-171
6.68.4 Function Qualifiers 6-171
6.68.5 Special Features and Restrictions 6-172
6.69 OA$FBT_INSTALL_LIBRARY 6-173
6.69.1 Syntax 6-173
6.69.2 Description 6-173
6.69.3 Function Parameters 6-173
6.69.4 Function Qualifiers 6-174
6.69.5 Examples 6-174
6.69.6 Special Features and Restrictions 6-174
6.70 OA$FBT_LIST_LIBTREE 6-175
6.70.1 Syntax 6-175
6.70.2 Description 6-175
6.70.3 Function Parameters 6-175
6.70.4 Function Qualifiers 6-175
6.70.5 Examples 6-176
6.71 OA$FBT_LIST_TREE 6-177
6.71.1 Syntax 6-177
6.71.2 Description 6-177
6.71.3 Function Parameters 6-177
6.71.4 Function Qualifiers 6-177
6.71.5 Examples 6-178
6.72 OA$FBT_REMOVE_LIBRARY 6-179
6.72.1 Syntax 6-179
6.72.2 Description 6-179
6.72.3 Function Parameters 6-179
6.72.4 Function Qualifiers 6-179
6.72.5 Examples 6-180
6.72.6 Special Features and Restrictions 6-180
6.73 OA_$FBT_WRITE_LIBRARY 6-181
6.73.1 Syntax 6-181
6.73.2 Description 6-181
6.73.3 Function Parameters 6-181
6.73.4 Function Qualifiers 6-182
6.73.5 Examples 6-182
6.73.6 Special Features and Restrictions 6-182

6.74 OA$FLD_nnn 6-183
6.74.1 Syntax 6-183
6.74.2 Description 6-183
6.74.3 Terminator Functions 6-183
6.74.4 Examples 6-186
6.75 OA$FLD_NEXT_LIST 6-188
6.75.1 Syntax 6-188
6.75.2 Description 6-188
6.75.3 Function Parameters 6-188
6.75.4 Function Qualifiers 6-188
6.76 OA$FLD_RECOGNITION 6-189
6.76.1 Syntax 6-189
6.76.2 Description 6-189
6.76.3 Function Parameters 6-189
6.76.4 Examples 6-189
6.76.5 Special Features and Restrictions 6-189
6.77 OA$FLD_SCROLL_OFF 6-190
6.77.1 Syntax 6-190
6.77.2 Description 6-190
6.77.3 Function Parameters 6-190
6.77.4 Examples 6-190
6.78 OA$FLD_SEARCH 6-191
6.78.1 Syntax 6-191
6.78.2 Description 6-191
6.78.3 Function Parameters 6-191
6.78.4 Examples 6-191
6.78.5 Special Features and Restrictions 6-191
6.79 OA$FLO_CLEAR_VA 6-192
6.79.1 Syntax 6-192
6.79.2 Description 6-192
6.79.3 Function Parameters 6-192
6.79.4 Function Qualifiers 6-192
6.79.5 Special Features and Restrictions 6-192
6.80 OA$FLO_CLOSE_LIB 6-193
6.80.1 Syntax 6-193
6.80.2 Description 6-193
6.80.3 Function Parameters 6-193
6.80.4 Function Qualifiers 6-193
6.80.5 Special Features and Restrictions 6-194

6.81 OA$FLO_OPEN_LIB 6-195
6.81.1 Syntax 6-195
6.81.2 Description 6-195
6.81.3 Function Parameters 6-195
6.81.4 Function Qualifiers 6-195
6.81.5 Special Features and Restrictions 6-196
6.82 OA$FRM_SET_FIELD 6-197
6.82.1 Syntax 6-197
6.82.2 Description 6-197
6.82.3 Function Parameters 6-197
6.82.4 Function Qualifiers 6-197
6.83 OA$HAR_CHECK_DECCRT 6-198
6.83.1 Syntax 6-198
6.83.2 Description 6-198
6.83.3 Function Parameters 6-198
6.83.4 Function Qualifiers 6-198
6.83.5 Special Features and Restrictions 6-199
6.84 This section has been removed 6-200
6.85 OA$INI_GLOBALS 6-201
6.85.1 Syntax 6-201
6.85.2 Description 6-201
6.85.3 Function Parameters 6-201
6.85.4 Function Qualifiers 6-201
6.86 OA$INI_INITIALIZE 6-202
6.86.1 Syntax 6-202
6.86.2 Description 6-202
6.86.3 Function Parameters 6-202
6.86.4 Function Qualifiers 6-202
6.86.5 Examples 6-203
6.87 OA$INI_LOGICALS 6-204
6.87.1 Syntax 6-204
6.87.2 Description 6-204
6.87.3 Function Parameters 6-204
6.87.4 Function Qualifiers 6-204
6.88 OA$INI_OPEN_LIB 6-205
6.88.1 Syntax 6-205
6.88.2 Description 6-205
6.88.3 Function Parameters 6-205
6.88.4 Function Qualifiers 6-205

6.89 OA$INI_SET_DEFAULT_DIR 6-206
6.89.1 Syntax 6-206
6.89.2 Description 6-206
6.89.3 Function Parameters 6-206
6.89.4 Function Qualifiers 6-206
6.90 OA$INI_SUBPROCESS 6-207
6.90.1 Syntax 6-207
6.90.2 Description 6-207
6.90.3 Function Parameters 6-207
6.90.4 Function Qualifiers 6-207
6.90.5 Examples 6-208
6.91 OA$KEY_COMMAND 6-209
6.91.1 Syntax 6-209
6.91.2 Description 6-209
6.91.3 Function Parameters 6-209
6.91.4 Special Features and Restrictions 6-209
6.92 OA$LIST_xxx 6-210
6.92.1 Syntax 6-210
6.92.2 Description 6-210
6.92.3 Function Parameters 6-210
6.92.4 Function Qualifiers 6-210
6.92.5 Examples 6-211
6.92.6 Special Features and Restrictions 6-211
6.93 OA$LOG_DISPLAY 6-212
6.93.1 Syntax 6-212
6.93.2 Description 6-212
6.93.3 Function Parameters 6-212
6.93.4 Function Qualifiers 6-212
6.93.5 Special Features and Restrictions 6-213
6.94 OA$LOG_TO_FILE 6-214
6.94.1 Syntax 6-214
6.94.2 Description 6-214
6.94.3 Function Parameters 6-214
6.94.4 Function Qualifiers 6-215
6.94.5 Examples 6-215
6.95 OA$MENU_LEVEL_PUSH 6-216
6.95.1 Syntax 6-216
6.95.2 Description 6-216
6.95.3 Function Parameters 6-217
6.95.4 Function Qualifiers 6-217

6.96 OA$MENU_SHOW_STACK 6-218
6.96.1 Syntax 6-218
6.96.2 Description 6-218
6.96.3 Function Parameters 6-219
6.96.4 Function Qualifiers 6-219
6.97 OA$MRG_TTOUT 6-220
6.97.1 Syntax 6-220
6.97.2 Description 6-220
6.97.3 Function Parameters 6-220
6.97.4 Examples 6-221
6.98 OA$MSG_PURGE 6-222
6.98.1 Syntax 6-222
6.98.2 Description 6-222
6.98.3 Function Parameters 6-222
6.98.4 Examples 6-222
6.99 OA$MSG_TRAP 6-223
6.99.1 Syntax 6-223
6.99.2 Description 6-223
6.99.3 Function Parameters 6-223
6.99.4 Function Qualifiers 6-223
6.100 OA$SCL_nnn 6-224
6.100.1 Syntax 6-224
6.100.2 Description 6-224
6.100.3 Scroll Functions 6-224
6.100.4 File Conversion for /SCROLL 6-227
6.100.5 Special Features and Restrictions 6-227
6.101 OA$SCP_STACK_DUMP 6-228
6.101.1 Syntax 6-228
6.101.2 Description 6-228
6.101.3 Function Parameters 6-228
6.101.4 Function Qualifiers 6-228
6.102 OA$STAT_GET_STATISTICS 6-229
6.102.1 Syntax 6-229
6.102.2 Description 6-229
6.102.3 Function Parameters 6-230
6.102.4 Function Qualifiers 6-230
6.102.5 Special Features and Restrictions 6-230
6.103 OA$SUB_CLOSE 6-231
6.103.1 Syntax 6-231
6.103.2 Description 6-231
6.103.3 Function Parameters 6-231
6.103.4 Function Qualifiers 6-231

6.104 OA$SUB_OPEN 6-232
6.104.1 Syntax 6-232
6.104.2 Description 6-232
6.104.3 Function Parameters 6-232
6.104.4 Function Qualifiers 6-232
6.105 OA$SYM_CLOSE 6-233
6.105.1 Syntax 6-233
6.105.2 Description 6-233
6.105.3 Function Parameters 6-233
6.105.4 Function Qualifiers 6-233
6.106 OA$SYM_GET_SYMBOL 6-234
6.106.1 Syntax 6-234
6.106.2 Description 6-234
6.106.3 Function Parameters 6-234
6.106.4 Function Qualifiers 6-235
6.106.5 Examples 6-235
6.107 OA$SYM_MAKE_SYMBOL 6-236
6.107.1 Syntax 6-236
6.107.2 Description 6-236
6.107.3 Function Parameters 6-236
6.107.4 Function Qualifiers 6-236
6.107.5 Examples 6-237
6.108 OA$SYM_OPEN 6-238
6.108.1 Syntax 6-238
6.108.2 Description 6-238
6.108.3 Function Parameters 6-238
6.108.4 Function Qualifiers 6-238
6.108.5 Special Features and Restrictions 6-239
6.109 OA$TRA_OUT 6-240
6.109.1 Syntax 6-240
6.109.2 Description 6-240
6.109.3 Function Parameters 6-240
6.109.4 Function Qualifiers 6-240
6.110 OA$TRA_SET 6-241
6.110.1 Syntax 6-241
6.110.2 Description 6-241
6.110.3 Function Parameters 6-241
6.110.4 Function Qualifiers 6-242
6.110.5 Examples 6-242
6.110.6 Special Features and Restrictions 6-243

6.111 OA$TRM_BRDCST_INIT 6-244
6.111.1 Syntax 6-244
6.111.2 Description 6-244
6.111.3 Function Parameters 6-244
6.111.4 Function Qualifiers 6-244
6.111.5 Examples 6-244
6.111A OA$TRM_BRDCST_STOP 6-244.2
6.111A.1 Syntax 6-244.2
6.111A.2 Description 6-244.2
6.111A.3 Function Parameters 6-244.2
6.111A.4 Function Qualifiers 6-244.2
6.111A.5 Examples 6-244.3
6.112 OA$TRM_INIT 6-245
6.112.1 Syntax 6-245
6.112.2 Description 6-245
6.112.3 Function Parameters 6-245
6.112.4 Function Qualifiers 6-245
6.113 OA$TRM_INQUIRE 6-246
6.113.1 Syntax 6-246
6.113.2 Description 6-246
6.113.3 Function Parameters 6-246
6.113.4 Function Qualifiers 6-246
6.114 OA$TRM_PORT_LINE 6-247
6.114.1 Syntax 6-247
6.114.2 Description 6-247
6.114.3 Function Parameters 6-247
6.114.4 Function Qualifiers 6-247
6.115 OA$TRM_PORT_LOG_ON 6-248
6.115.1 Syntax 6-248
6.115.2 Description 6-248
6.115.3 Function Parameters 6-248
6.115.4 Function Qualifiers 6-248
6.116 OA$TRM_PORT_OFF 6-249
6.116.1 Syntax 6-249
6.116.2 Description 6-249
6.116.3 Function Parameters 6-249
6.116.4 Function Qualifiers 6-249
6.117 OA$TRM_PORT_ON 6-250
6.117.1 Syntax 6-250
6.117.2 Description 6-250
6.117.3 Function Parameters 6-250
6.117.4 Function Qualifiers 6-250
6.118 OA$TRM_PORT_PRINT 6-251
6.118.1 Syntax 6-251
6.118.2 Description 6-251

6.118.3 Function Parameters 6-251
6.118.4 Function Qualifiers 6-251
6.119 OA$TXL_CLOSE 6-252
6.119.1 Syntax 6-252
6.119.2 Description 6-252
6.119.3 Function Parameters 6-252
6.119.4 Function Qualifiers 6-252
6.119.5 Special Features and Restrictions 6-252
6.120 OA$TXL_COMPILE 6-253
6.120.1 Syntax 6-253
6.120.2 Description 6-253
6.120.3 Function Parameters 6-253
6.120.4 Function Qualifiers 6-253
6.121 OA$TXL_DIRECTORY 6-254
6.121.1 Syntax 6-254
6.121.2 Description 6-254
6.121.3 Function Parameters 6-254
6.121.4 Function Qualifiers 6-254
6.122 OA$TXL_INSTALL 6-255
6.122.1 Syntax 6-255
6.122.2 Description 6-255
6.122.3 Function Parameters 6-255
6.122.4 Function Qualifiers 6-255
6.122.5 Special Features and Restrictions 6-255
6.123 OA$TXL_LIST 6-256
6.123.1 Syntax 6-256
6.123.2 Description 6-256
6.123.3 Function Parameters 6-256
6.123.4 Function Qualifiers 6-256
6.123.5 Examples 6-256
6.124 OA$TXL_OPEN 6-257
6.124.1 Syntax 6-257
6.124.2 Description 6-257
6.124.3 Function Parameters 6-257
6.124.4 Function Qualifiers 6-257
6.124.5 Special Features and Restrictions 6-258
6.125 OA$TXL_REMOVE 6-259
6.125.1 Syntax 6-259
6.125.2 Description 6-259
6.125.3 Function Parameters 6-259
6.125.4 Function Qualifiers 6-259
6.125.5 Special Features and Restrictions 6-259

6.126 OA$VAL_SET_VALID 6-260
6.126.1 Syntax 6-260
6.126.2 Description 6-260
6.126.3 Function Parameters 6-260
6.126.4 Function Qualifiers 6-260
6.126.5 Examples 6-261
6.127 PARSE_USER 6-262
6.127.1 Syntax 6-262
6.127.2 Description 6-262
6.127.3 Function Parameters 6-262
6.127.4 Function Qualifiers 6-263
6.127.5 Examples 6-263
6.128 PORT_ON, PORT_OFF 6-264
6.128.1 Syntax 6-264
6.128.2 Description 6-264
6.128.3 Function Parameters 6-264
6.128.4 Function Qualifiers 6-264
6.128.5 Examples 6-264
6.128.6 Special Features and Restrictions 6-264
6.129 PRINT_SCREEN 6-265
6.129.1 Syntax 6-265
6.129.2 Description 6-265
6.129.3 Function Parameters 6-266
6.129.4 Function Qualifiers 6-266
6.129.5 Examples 6-266
6.130 PROMPT 6-267
6.130.1 Syntax 6-267
6.130.2 Description 6-267
6.130.3 Function Parameters 6-268
6.130.4 Function Qualifiers 6-268
6.130.5 Examples 6-268
6.130.6 Special Features and Restrictions 6-269
6.131 PROMPT_ID 6-271
6.131.1 Syntax 6-271
6.131.2 Description 6-271
6.131.3 Function Parameters 6-272
6.131.4 Function Qualifiers 6-272
6.132 PURGE_FILE 6-273
6.132.1 Syntax 6-273
6.132.2 Description 6-273
6.132.3 Function Parameters 6-273
6.132.4 Function Qualifiers 6-274
6.132.5 Examples 6-274
6.132.6 Special Features and Restrictions 6-275

6.133 PUT_FIELD 6-277
6.133.1 Syntax 6-277
6.133.2 Description 6-277
6.133.3 Function Parameters 6-277
6.133.4 Function Qualifiers 6-278
6.133.5 Examples 6-278
6.133.6 Special Features and Restrictions 6-278
6.134 QUEUE_BATCH 6-279
6.134.1 Syntax 6-279
6.134.2 Description 6-279
6.134.3 Function Parameters 6-279
6.134.4 Function Qualifiers 6-279
6.134.5 Examples 6-288
6.134.6 Special Features and Restrictions 6-282
6.135 QUEUE_PRINT 6-284
6.135.1 Syntax 6-284
6.135.2 Description 6-284
6.135.3 Function Parameters 6-284
6.135.4 Function Qualifiers 6-284
6.135.5 Examples 6-287
6.135.6 Special Features and Restrictions 6-288
6.136 RENAME 6-289
6.136.1 Syntax 6-289
6.136.2 Description 6-289
6.136.3 Function Parameters 6-289
6.136.4 Function Qualifiers 6-289
6.136.5 Special Features and Restrictions 6-290
6.137 REPEAT 6-291
6.137.1 Syntax 6-291
6.137.2 Description 6-291
6.137.3 Function Parameters 6-291
6.137.4 Function Qualifiers 6-291
6.137.5 Examples 6-291
6.137.6 Special Features and Restrictions 6-292
6.138 SAVE 6-293
6.138.1 Syntax 6-293
6.138.2 Description 6-293
6.138.3 Function Parameters 6-293
6.138.4 Function Qualifiers 6-294
6.138.5 Examples 6-294
6.138.6 Special Features and Restrictions 6-294

6.139 SCRIPT 6-296
6.139.1 Syntax 6-296
6.139.2 Description 6-296
6.139.3 Function Parameters 6-296
6.139.4 Function Qualifiers 6-297
6.139.5 Examples 6-297
6.139.6 Special Features and Restrictions 6-297
6.140 SET_DEFAULT 6-298
6.140.1 Syntax 6-298
6.140.2 Description 6-298
6.140.3 Function Parameters 6-298
6.140.4 Function Qualifiers 6-298
6.140.5 Examples 6-299
6.140.6 Special Features and Restrictions 6-299
6.141 SET_MENU 6-300
6.141.1 Syntax 6-300
6.141.2 Description 6-300
6.141.3 Function Parameters 6-300
6.141.4 Function Qualifiers 6-300
6.141.5 Examples 6-301
6.141.6 Special Features and Restrictions 6-301
6.142 SHOW_FORM 6-302
6.142.1 Syntax 6-302
6.142.2 Description 6-302
6.142.3 Function Parameters 6-302
6.142.4 Function Qualifiers 6-302
6.143 SORT_LIST 6-303
6.143.1 Syntax 6-303
6.143.2 Description 6-303
6.143.3 Function Parameters 6-303
6.143.4 Function Qualifiers 6-305
6.143.5 Examples 6-305
6.143.6 Special Features and Restrictions 6-305
6.144 SPAWN 6-306
6.144.1 Syntax 6-306
6.144.2 Description 6-306
6.144.3 Function Parameters 6-306
6.144.4 Function Qualifiers 6-307
6.144.5 Special Features and Restrictions 6-307

6.145 STATISTICS 6-308
6.145.1 Syntax 6-308
6.145.2 Description 6-308
6.145.3 Function Parameters 6-308
6.145.4 Function Qualifiers 6-309
6.145.5 Examples 6-309
6.146 TEST_SPEC 6-310
6.146.1 Syntax 6-310
6.146.2 Description 6-310
6.146.3 Function Parameters 6-310
6.146.4 Function Qualifiers 6-310
6.146.5 Examples 6-311
6.146.6 Special Features and Restrictions 6-311
6.147 TIME 6-312
6.147.1 Syntax 6-312
6.147.2 Description 6-312
6.147.3 Function Parameters 6-312
6.147.4 Function Qualifiers 6-312
6.147.5 Examples 6-313
6.148 UNWIND 6-314
6.148.1 Syntax 6-314
6.148.2 Description 6-314
6.148.3 Function Parameters 6-314
6.148.4 Function Qualifiers 6-314
6.148.5 Examples 6-315
6.148.6 Special Features and Restrictions 6-315
6.149 VERSION 6-316
6.149.1 Syntax 6-316
6.149.2 Description 6-316
6.149.3 Function Parameters 6-316
6.149.4 Function Qualifiers 6-316
6.149.5 Examples 6-317
6.150 WAIT 6-318
6.150.1 Syntax 6-318
6.150.2 Description 6-318
6.150.3 Function Parameters 6-318
6.150.4 Function Qualifiers 6-319
6.150.5 Examples 6-319
6.150.6 Special Features and Restrictions 6-320

6.151 WPSPLUS$CBI_CLOSE_WINDOW 6-321
6.151.1 Syntax 6-321
6.151.2 Description 6-321
6.151.3 Function Parameters 6-321
6.151.4 Function Qualifiers 6-321
6.151.5 Examples 6-321
6.151.6 Special Features and Restrictions 6-321
6.152 WPSPLUS$CBI_CLOSE_WINDOW2 6-322
6.152.1 Syntax 6-322
6.152.2 Description 6-322
6.152.3 Function Parameters 6-322
6.152.4 Function Qualifiers 6-322
6.152.5 Examples 6-322
6.152.6 Special Features and Restrictions 6-322
6.153 WPSPLUS$CBI_OPEN_WINDOW 6-323
6.153.1 Syntax 6-323
6.153.2 Description 6-323
6.153.3 Function Parameters 6-323
6.153.4 Function Qualifiers 6-323
6.153.5 Examples 6-323
6.153.6 Special Features and Restrictions 6-324
6.154 WPSPLUS$CBI_OPEN_WINDOW2 6-325
6.154.1 Syntax 6-325
6.154.2 Description 6-325
6.154.3 Function Parameters 6-325
6.154.4 Function Qualifiers 6-325
6.154.5 Examples 6-326
6.154.6 Special Features and Restrictions 6-326
6.155 WPSPLUS_SPELL 6-327
6.155.1 Syntax 6-327
6.155.2 Description 6-327
6.155.3 Function Parameters 6-327
6.155.4 Function Qualifiers 6-327
6.155.5 Examples 6-327
6.155.6 Special Features and Restrictions 6-327
6.156 WRITE 6-328
6.156.1 Syntax 6-328
6.156.2 Description 6-328
6.156.3 Function Parameters 6-328
6.156.4 Function Qualifiers 6-329
6.156.5 Examples 6-329
6.156.6 Special Features and Restrictions 6-329

6.157 YESNO_PROMPT 6-330
6.157.1 Syntax 6-330
6.157.2 Description 6-330
6.157.3 Function Parameters 6-330
6.157.4 Function Qualifiers 6-331
6.157.5 Examples 6-331
6.157.6 Special Features and Restrictions 6-332
6.158 YESNO_PROMPT_ID 6-333
6.158.1 Syntax 6-333
6.158.2 Description 6-333
6.158.3 Function Parameters 6-333
6.158.4 Function Qualifiers 6-334
6.158.5 Special Features and Restrictions 6-334

Volume 3: Applications

Preface

7 The Subprocess

7.1 Introduction 7-1
7.2 Initializing the Subprocess 7-1
7.3 Communicating with the Subprocess Through Mailboxes 7-2
7.3.1 SYS$INPUT 7-4
7.3.2 OAMAILBOX and DCLMAILBOX 7-4
7.4 WRITE OAMAILBOX Syntax 7-6
7.5 Defining and Using Symbols 7-8
7.5.1 Creating DCL Symbols with /COPY and /PUT_SAVE 7-9
7.5.2 Creating DCL Symbols with Functions 7-10
7.6 Common DCL Symbols 7-12
7.6.1 Passing Command-Procedure Parameters to the Subprocess from ALL-IN-1 7-16
7.7 ALL-IN-1 Logicals 7-16
7.8 Running Command Procedures and Programs 7-19
7.8.1 Running Command Procedures 7-19
7.8.2 Obtaining User Input in a Program or Command Procedure 7-20

8 ALL-IN-1 Symbols

8.1 Introduction 8-1
8.2 Basic Symbol Types 8-2
8.2.1 Quoted String 8-2
8.2.2 Field Name 8-2
8.2.3 Permanent Symbol Table 8-2
8.2.4 Special Symbols 8-3
8.2.5 Local Symbol Table 8-4
8.2.6 Data Set References 8-4

8.3 Using Data Set References 8-5
8.3.1 Combining and Nesting DSRs 8-6
8.3.2 Specifying Alternate Keys with DSRs 8-6
8.3.3 Special Field Names 8-7
8.4 Symbol Substrings 8-10
8.5 Symbol Lists 8-11
8.6 Iterative Symbol Substitution 8-12
8.7 Specifying Date/Time Values 8-13
8.7.1 Creating and Updating Symbols and Data Sets 8-14
8.7.2 Getting Values from Symbols 8-15
8.7.3 Values Stored in Multiple Symbols 8-15
8.8 Special Symbols 8-15
8.9 Permanent Symbols 8-42

9 The File Cabinet

9.1 File Cabinet Concepts 9-2
9.2 File Cabinet Organization 9-4
9.2.1 OAUSER:DOCDB.DAT, the Personal File Cabinet Index 9-4
9.3 Document Attributes 9-8
9.3.1 The Document Attributes Files 9-9
9.3.2 Document Attributes 9-9
9.3.3 The System Pending File 9-12
9.4 Using the File Cabinet 9-13
9.4.1 Summary of CABINET Subfunctions 9-13
9.4.2 Accessing File Cabinet Data 9-15
9.4.3 The Shared Area 9-19
9.4.4 Shared Libraries 9-19
9.5 CABINET Subfunctions 9-20

10 Text Processing

10.1 Overview 10-1
10.2 Text Data Sets 10-2
10.3 Creating Files with the EDIT function 10-3
10.4 Features of the Default Editor 10-4
10.4.1 Editor Modes 10-4
10.4.2 Communicating with the Rest of ALL-IN-1 Through the Editor 10-10
10.4.3 Moving Within a Document 10-13
10.4.4 Moving Text Within a Document 10-14
10.4.5 Including Outside Text into a Document 10-16
10.4.6 Creating New Files While in the Editor 10-17
10.4.7 Customizing the Default Editor 10-18

10.5 Features of the WPS-PLUS Editor 10-21
10.6 List Processing 10-21
10.7 Formatting and Outputting Text 10-28

11 Electronic Messaging

11.1 Introduction 11-1
11.2 Electronic Messaging System Concepts 11-1
11.2.1 Current Message 11-1
11.2.2 Message Header and Text 11-2
11.2.3 Mail Interfaces 11-2
11.2.4 Mail Folders 11-2
11.2.5 Nicknames and Distribution Lists 11-3
11.2.6 Mail Delivery Mechanism 11-3
11.2.7 Alternative Message Transport Systems (MTS) 11-5
11.3 Special Symbols 11-9
11.4 Customizing the Message Header and Show-Message Displays 11-9
11.5 MAIL Subfunctions 11-10
11.5.1 ANSWER 11-11
11.5.2 ATTACH 11-13
11.5.3 AUTO_FORWARD 11-13
11.5.4 AUTO_REPLY 11-14
11.5.5 CANCEL 11-14
11.5.6 CANCEL_AUTO_REPLY 11-15
11.5.7 CC 11-15
11.5.8 CLOSE_MESSAGE 11-16
11.5.9 CREATE 11-16.1
11.5.10 DEFAULT_MESSAGE 11-19
11.5.11 DEFER 11-20
11.5.12 DELETE 11-20
11.5.13 DELIVERY_RECEIPT 11-21
11.5.14 DETACH 11-22
11.5.15 DIST_COPY 11-22
11.5.16 DIST_CREATE 11-23
11.5.17 DIST_DELETE 11-23
11.5.18 DIST_EDIT 11-24
11.5.19 DIST_LIST 11-24
11.5.20 EDIT 11-25
11.5.21 EXCHANGE_CURMES 11-26
11.5.22 EXPAND_DIST_LIST 11-26
11.5.23 FILE_ATTACHMENT 11-27

11.5.24 FILE_BODY 11-28
11.5.25 FILE_MESSAGE 11-29
11.5.26 FORWARD 11-30
11.5.27 FORWARDABLE 11-32
11.5.28 FUNCTION 11-32
11.5.29 GET 11-33
11.5.30 INDEX 11-33
11.5.31 MAIL_DOCUMENT 11-34
11.5.32 NEXT 11-36
11.5.33 NO_DELIVERY_RECEIPT 11-37
11.5.34 NON_FORWARDABLE 11-37
11.5.35 ORIGINAL 11-37
11.5.36 OUTBOX 11-38
11.5.37 POP_CURMES 11-38
11.5.38 PRINT 11-39
11.5.39 PRIORITY 11-40
11.5.40 PUSH_CURMES 11-41
11.5.41 READ 11-42
11.5.42 READ_RECEIPT 11-43
11.5.43 SEND 11-44
11.5.44 SHOW 11-46
11.5.45 STATUS 11-46
11.5.46 SUBJECT 11-47
11.5.47 TEXT 11-48
11.5.48 TO 11-48
11.5.49 VAXMAIL_CHECK 11-49
11.6 Electronic Messaging Data Set References 11-50

12 External Communications

12.1 Overview 12-1
12.2 Multinode ALL-IN-1 Applications 12-1
12.3 The Communications Control Subsystem 12-2
12.3.1 Communications Control Concepts 12-2
12.3.2 CXP Symbols 12-2
12.3.3 CXP Subfunctions 12-5
12.3.4 CXP Subfunctions that Differ from Control Document Commands 12-9
12.3.5 Using Communication Control Commands 12-11
12.3.6 Example of Customizing Communications Control 12-12

13 Time Management

13.1 Introduction 13-1
13.2 Organization 13-1
13.2.1 Time Management Activities 13-2
13.2.2 Subfunctions and the Current CALENDAR Date/Time 13-2
13.2.3 CALENDAR Files 13-3
13.3 CALENDAR Subfunctions 13-3
13.3.1 CALENDAR ADD_ATTENDEE 13-3
13.3.2 CALENDAR CANCEL APPOINTMENT 13-4
13.3.3 CALENDAR CANCEL MEETING 13-4
13.3.4 CALENDAR CLEAR ATTENDEES 13-5
13.3.5 CALENDAR CONVERT APPOINTMENTS 13-5
13.3.6 CALENDAR CONVERT ACTION_ITEMS 13-5
13.3.7 CALENDAR CONVERT MEETINGS 13-6
13.3.8 CALENDAR DISPLAY ANSWER 13-6
13.3.9 CALENDAR DISPLAY APPOINTMENT 13-7
13.3.10 CALENDAR DISPLAY GRAPH 13-8
13.3.11 CALENDAR DISPLAY MEETING 13-9
13.3.12 CALENDAR DISPLAY MONTH 13-9
13.3.13 CALENDAR DISPLAY NEXT APPOINTMENT 13-10
13.3.14 CALENDAR DISPLAY NEXT MEETING 13-11
13.3.15 CALENDAR DISPLAY NEXT REPLY 13-11
13.3.16 CALENDAR DISPLAY REPLY 13-12
13.3.17 CALENDAR INITIALIZE FILES 13-13
13.3.18 CALENDAR INITIALIZE MONTH 13-13
13.3.19 CALENDAR MAIL_TO_ATTENDEES 13-14
13.3.20 CALENDAR MAKE_ATTENDEE_SYMBOLS 13-14
13.3.21 CALENDAR MAKE_MEETING_SYMBOLS 13-15
13.3.22 CALENDAR PRINT ALL 13-16
13.3.23 CALENDAR PRINT CONFLICTS 13-16
13.3.24 CALENDAR PRINT CUSTOM 13-17
13.3.24A CALENDAR PRINT DAY_REMIND 13-17
13.3.25 CALENDAR PRINT DAY_SCHEDULE 13-18
13.3.26 CALENDAR PRINT MONTH 13-18
13.3.26A CALENDAR PRINT TODO 13-19
13.3.27 CALENDAR PRINT TWOCAL 13-19
13.3.28 CALENDAR PRINT WEEK_REMINDERS 13-20
13.3.29 CALENDAR PRINT WEEK_SCHEDULE 13-20
13.3.30 CALENDAR PURGE 13-20.1

13.3.31 CALENDAR REMOVE_ATTENDEES 13-20.1
13.3.32 CALENDAR REPLY_TO_MEETING 13-21
13.3.33 CALENDAR RESCHEDULE APPOINTMENT 13-21
13.3.34 CALENDAR RESCHEDULE MEETING 13-22
13.3.35 CALENDAR SCAN 13-22
13.3.36 CALENDAR SCHEDULE APPOINTMENT 13-23
13.3.37 CALENDAR SCHEDULE MEETING 13-23
13.3.38 CALENDAR SET ALTERNATE_USER 13-24
13.3.39 CALENDAR SET DATE 13-25
13.3.40 CALENDAR SET NETWORK OFF 13-26
13.3.41 CALENDAR SET NETWORK ON 13-26
13.3.42 CALENDAR VERSION 13-27
13.4 Calendar File Layouts 13-27
13.4.1 User File Layout 13-27
13.4.2 Meeting File Layout 13-29
13.4.3 Attendee File Layout 13-30
13.5 CALENDAR Data Sets 13-30

14 The Help and Message Facilities

14.1 Introduction 14-1
14.2 Help Facility 14-1
14.2.1 The HELP Function 14-1
14.2.2 The Help Library 14-6
14.2.3 Maintaining, Updating, and Switching Between Help Libraries 14-18
14.2.4 Help at a Prompt 14-22
14.3 Message Facility 14-22
14.3.1 The Message File 14-23
14.3.2 Updating the Message File 14-23
14.3.3 Message Severity and the Message Buffer 14-24
14.3.4 Help on Error Messages 14-26

15 The Script Facility

15.1 Overview 15-1
15.2 Scripts 15-1
15.2.1 Script Modes 15-2
15.2.2 Interactive Directives and the Pseudo-Input Buffer 15-3
15.2.3 Exiting from Scripts 15-4
15.2.4 ALL-IN-1 TXLs 15-5
15.2.5 Script-File Format 15-7

15.3 Script Directives 15-9
15.4 Script Directives as Functions 15-31
15.5 Sample Script 15-32
15.6 User Defined Processes 15-33
15.7 Running ALL-IN-1 Indirectly or in Batch Mode 15-33
15.8 Special Features and Restrictions 15-33
15.8.1 Creating a Script with a Script 15-33
15.8.2 UDPs for the Desk Calculator 15-34

16 Application Development Tools and Design Guidelines

16.1 Introduction 16-1
16.1.1 Where to Start 16-2
16.2 The Forms Development Subsystem 16-2
16.2.1 Using Form Libraries 16-9
16.2.2 Testing Newly Created Forms 16-11
16.3 The Programming Functions Subsystem 16-12
16.4 Other Development Tools 16-14
16.4.1 Debugging Features 16-15
16.4.2 Using the Subprocess Interactively 16-15
16.4.3 Using the Editor 16-16
16.4.4 Using the GET Function 16-17
16.4.5 Using View Forms 16-18
16.5 Application Design Guidelines 16-18
16.5.1 Standard Applications 16-19
16.5.2 Standalone Applications 16-20
16.5.3 User-Defined Functions 16-21
16.6 Performance Considerations 16-22
16.7 Documenting a Customized System 16-23
16.7.1 User Documentation 16-23
16.7.2 Help Documentation 16-24

A Exercises

A.1 Introduction A-1
A.2 Exercise 1 — Using Standard ALL-IN-1 Subsystems A-2
A.2.1 Objective A-2
A.2.2 Preparation A-2
A.2.3 Exercise A-2
A.2.4 Summary A-3
A.2.5 References A-3

A.3 Exercise 2 — Modifying the User Profile A-3
A.3.1 Objective A-3
A.3.2 Preparation A-3
A.3.3 Exercise A-3
A.3.4 Summary A-4
A.3.5 References A-4
A.4 Exercise 3 — Advanced User Tasks A-4
A.4.1 Objective A-4
A.4.2 Preparation A-4
A.4.3 Exercise A-4
A.4.4 Summary A-5
A.4.5 References A-5
A.5 Exercise 4 — Modifying the User Profile (Programmer Level) A-5
A.5.1 Objective A-5
A.5.2 Preparation A-6
A.5.3 Exercise A-6
A.5.4 Summary A-6
A.5.5 References A-7
A.6 Exercise 5 — Accessing the Subprocess A-7
A.6.1 Objective A-7
A.6.2 Preparation A-7
A.6.3 Exercise A-7
A.6.4 Summary A-8
A.6.5 References A-8
A.7 Exercise 6 — Using Functions Interactively A-8
A.7.1 Objective A-8
A.7.2 Preparation A-8
A.7.3 Exercise A-8
A.7.4 Summary A-9
A.7.5 References A-9
A.8 Exercise 7 — Using the Hard Copy Interface A-9
A.8.1 Objective A-9
A.8.2 Preparation A-9
A.8.3 Exercise A-10
A.8.4 Summary A-11
A.8.5 References A-11
A.9 Exercise 8 — Using Development Tools A-11
A.9.1 Objective A-11
A.9.2 Preparation A-11
A.9.3 Exercise A-11
A.9.4 Summary A-12
A.9.5 References A-12

A.10 Exercise 9 — Modifying a Menu A-12
A.10.1 Objective A-12
A.10.2 Preparation A-12
A.10.3 Exercises A-12
A.10.4 Summary A-13
A.10.5 References A-13
A.11 Exercise 10 — Creating an Entry Form for an Indexed File A-13
A.11.1 Objective A-13
A.11.2 Preparation A-14
A.11.3 Exercise A-14
A.11.4 Summary A-15
A.11.5 References A-15
A.12 Exercise 11 — Creating a Select Form A-15
A.12.1 Objective A-15
A.12.2 Preparation A-15
A.12.3 Exercise A-15
A.12.4 Summary A-16
A.12.5 References A-16
A.13 Exercise 12 — Modifying a Script A-16
A.13.1 Objective A-16
A.13.2 Preparation A-17
A.13.3 Exercise A-17
A.13.4 Summary A-17
A.13.5 References A-17
A.13.6 Answers A-18
A.14 Exercise 13 — Creating a Shared Template A-19
A.14.1 Objective A-19
A.14.2 Preparation A-19
A.14.3 Exercise A-19
A.14.4 Summary A-20
A.14.5 References A-20
A.15 Exercise 14 — Using DATATRIEVE A-20
A.15.1 Objective A-20
A.15.2 Preparation A-21
A.15.3 Exercise A-21
A.15.4 Summary A-22
A.15.5 References A-22

B ALL-IN-1 Hard Copy

B.1 ALL-IN-1 Hard Copy B-1
B.1.1 Intended Use B-1
B.1.2 Terminal Support B-2

B.1.3 Function Keys B-3
B.1.4 Appearance on the Screen B-3
B.1.5 /HARD Qualifiers B-4
B.1.6 Function-Key Entry B-5
B.1.7 Field-Full Condition B-5

C Data Sets

C.1 Overview C-1
C.2 Special Data Sets C-2
C.2.1 OA$DATE C-2
C.2.2 OA$DIR C-3
C.2.3 OA$DIR_PARSE C-6
C.2.4 OA$EDITOR C-7
C.2.5 OA$MAIDES C-7
C.2.6 OA$MAIL_ADDRESS C-8
C.2.7 OA$MAILPRIO C-8
C.2.8 OA$TABLE C-8
C.2.9 OA$YESNO C-9
C.2.10 OA$YN C-9
C.3 Calendar Validation Data Sets C-9
C.3.1 CAL$ATTENDEE C-9
C.3.2 CAL$CALENDAR C-10
C.3.3 CAL$END_TIME_CHECK:key C-10
C.3.4 CAL$MEETING C-10
C.3.5 CAL$MTG_TIME_CHECK C-10
C.3.6 CAL$SET_DATE C-10
C.3.7 CAL$SET_OWNER C-11
C.3.8 CAL$SET_SECOND C-11
C.3.9 CAL$TIME_CHECK C-11
C.4 Calendar Scrolling Data Sets C-11
C.4.1 CAL$SCROLL_BOTH C-12
C.4.2 CAL$SCROLL_CONFLICTS C-12
C.4.3 CAL$SCROLL_CUSTOM C-12
C.4.4 CAL$SCROLL_DAY C-12
C.4.5 CAL$SCROLL_TWOCAL C-12
C.4.6 CAL$SCROLL_WEEK_REM C-13
C.4.7 CAL$SCROLL_WEEK_SCH C-13
C.5 General Scrolling Data Sets C-13
C.5.1 SCROLL$FUNCTION C-13
C.6 Data Set Statistics C-13

D Mnemonics for Script Input

Glossary

Figures

How To Use this APR xlv
ALL-IN-1 Bookshelf xlvi
1-1 The VAX Information Architecture 1-2
1-2 ALL-IN-1 and the VAX Information Architecture 1-4
1-3 Form and Function 1-5
2-1 Primary ALL-IN-1 Flow-Control Framework 2-4
2-2 Flow Control Path: Display Main Menu 2-6
3-1 Menu Processing Flow Control 3-30
3-2 Form WPSEARCH and assumed Named Data 3-62
5-1 A Subsystem Menu 5-2
5-2 A Select Form 5-4
5-3 A Create Form 5-5
5-4 A Secondary Menu Form 5-8
7-1 Flow Control Between the Subprocess and Main Process 7-3
8-1 Obtaining Data with a DSR 8-9
9-1 File Cabinet Organization 9-13
10-1 Key Numbers for the Default Editor 10-12
10-2 Entry Form FORMAT (Document Handling and File Processing) 10-29
16-1 Forms Development Menu 16-3
16-2 Programming Functions Menu 16-13
16-3 Level-1 HELP Module Template 16-28
16-4 Level-2 HELP Module Template 16-29
16-5 Level-3 HELP Module Template 16-29

Tables

3-1a Form AIE: Directives 3-40
3-1b Form AIE: Field Information 3-41
3-2 File OA$LIB:ACTITEM.FDL 3-43
4-1 SCROLL$FUNCTION Actions 4-37
4-2 Control-Function Calls to Internal Terminator Functions 4-67
4-3 Terminator Functions, Dispositions, and OA$FORM_DISPOSE 4-68
4-4 OA$FORM_TERMINATOR Values 4-69

6-1 COMPUTE Operators and Operands 6-38
6-2 DATE_CONVERT Parameters 6-49
6-3 Relational Operators 6-85
6-4 Boolean Expressions 6-86.1
6-5 Recognized Editor Styles 6-131
7-1 ALL-IN-1 Subprocess Logicals 7-18
8-1 Special Symbols 8-16
8-2 Permanent Symbols 8-42
9-1a Document Attributes 9-10
9-1b Extended Attributes 9-11
10-1 Recognized Editor Styles 10-3
10-2 File Processors 10-32
B-1 Supported Terminals B-2
C-1 Data Set Statistics (OA$DSA_CACHE_STATUS output) C-14

Preface

Purpose of this Guide

The *ALL-IN-1 Office Menu Application Programmer's Reference* (APR) gives a detailed overview of ALL-IN-1 and presents the tools available to the applications programmer for modifying or adding new applications. The APR also provides Application Design Guidelines and Exercises.

Intended Audience

The APR is written for the applications programmer familiar with VAX/VMS and its layered products. References to other documents that provide additional information are presented in the ALL-IN-1 Bookshelf illustrated in this Preface. References are also noted, where relevant, throughout this APR.

Before creating an application, you should be able to:

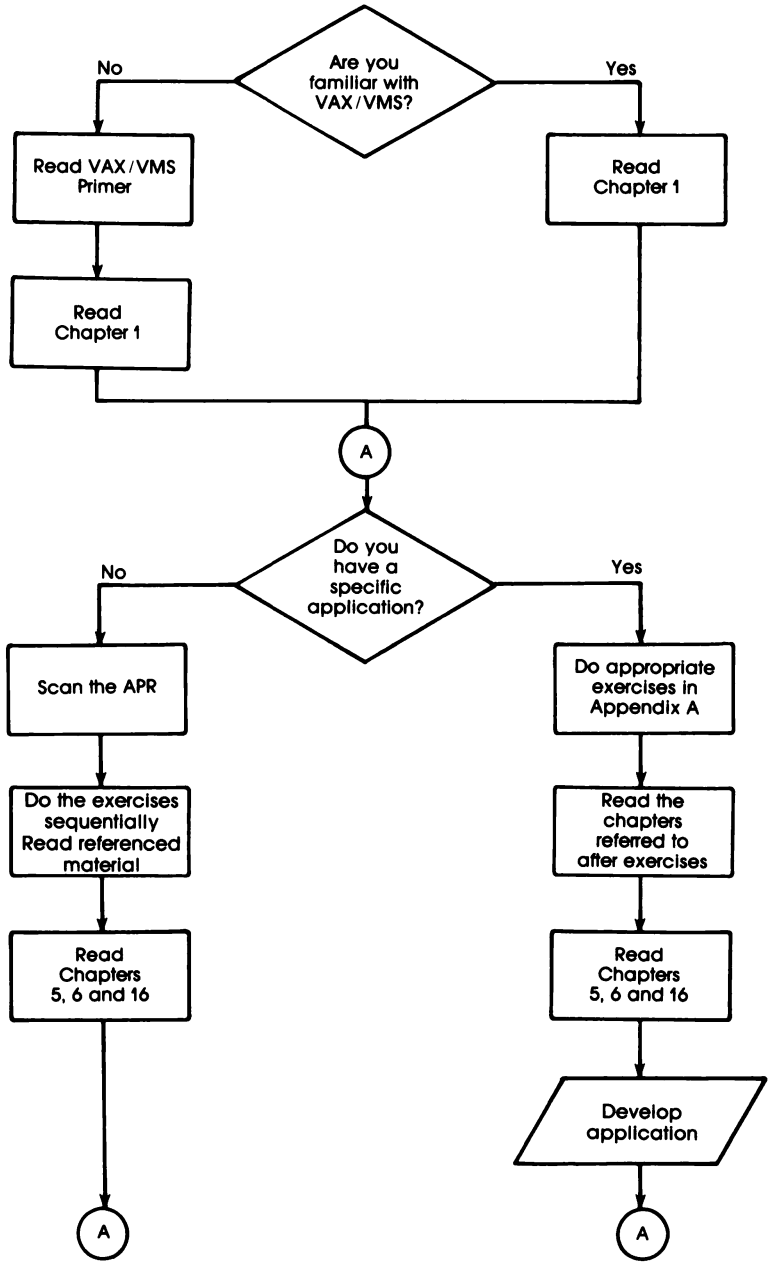
- Understand the following VAX/VMS concepts:
  - DCL
  - directory
  - file type
  - process
  - protection
  - shareable image

  - library — subprocess
  - logical — symbol
  - privilege — user identification code
- Login to a VMS system.
- Edit a file using the EDT text editor.
- Create a simple DIGITAL Command Language (DCL) command procedure.
- Create a Forms Management System (FMS) form, and understand the meaning and use of Named Data within a form.
- Create and access subdirectories.
- Create DCL symbols and assign logical names.

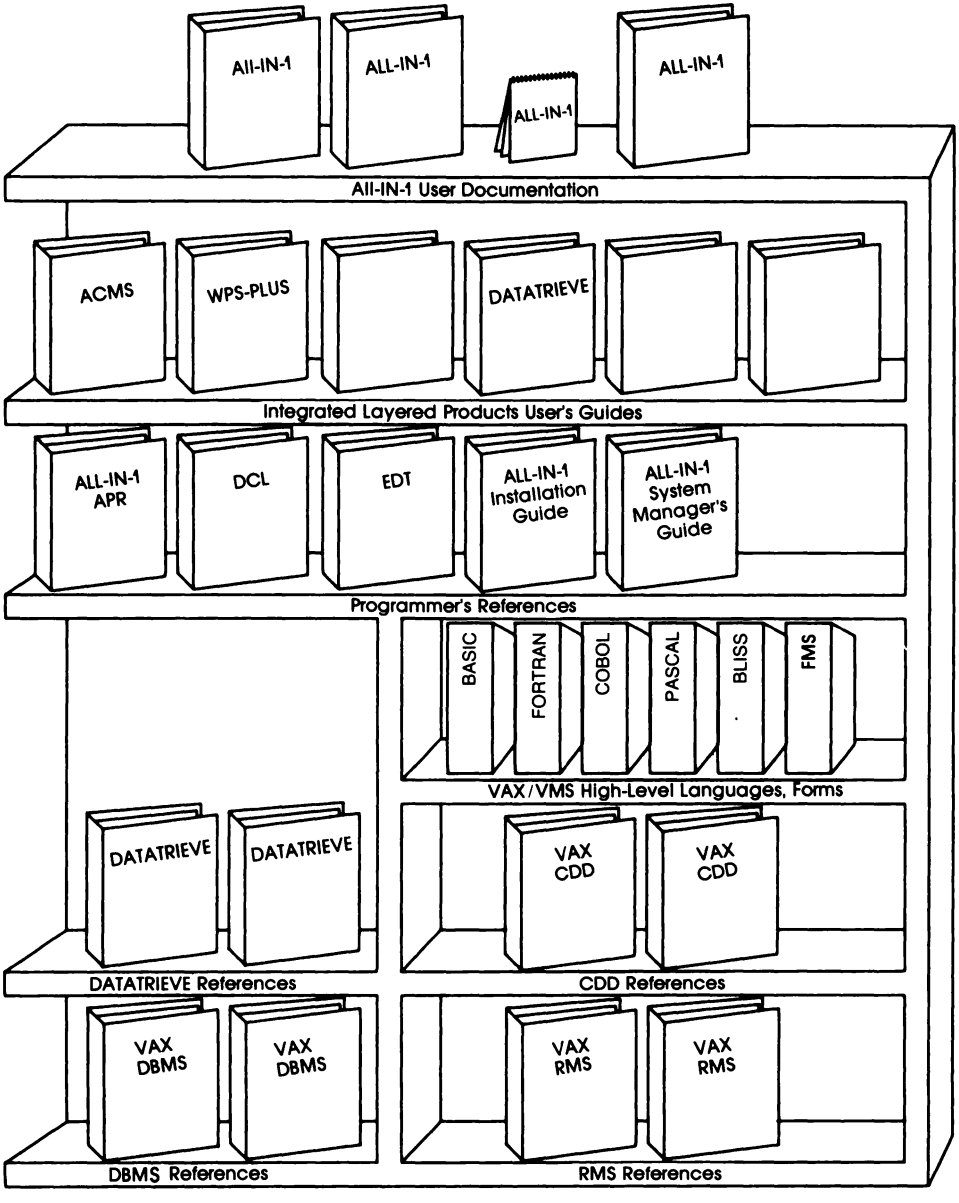
If you are unfamiliar with these concepts, refer to the flowchart on the following page and to the Glossary at the end of this guide. The flowchart may also be useful if you are familiar with VAX/VMS concepts.

How the APR is Organized

The APR has three volumes: Volume 1, Flow Control, covers the kernel of ALL-IN-1. Volume 2, ALL-IN-1 Functions, describes the functions. These are calls to ALL-IN-1 facilities to request that a specific action be performed. Volume 3, Applications, covers those closely integrated applications that use the Flow Control facilities.


Are you familiar with VAX/VMS?
No
Yes
Read VAX/VMS Primer
Read Chapter 1
Read Chapter 1
A
Do you have a specific application?
No
Yes
Scan the APR
Do appropriate exercises in Appendix A
Do the exercises sequentially Read referenced material
Read the chapters referred to after exercises
Read Chapters 5, 6 and 16
Read Chapters 5, 6 and 16
Develop application
A
A

MK-02000-00



ALL-IN-1 Bookshelf

This Bookshelf contains all the documents referenced throughout the APR. These documents are also listed in the Related Documents section that follows.

Related Documents

The following documents are part of the ALL-IN-1 Office Menu Documentation Set:

User Documentation

ALL-IN-1 Office Menu Getting Started Guide
ALL-IN-1 Office Menu User's Reference
Volume 1
Volume 2
ALL-IN-1 Office Menu Quick Lookup

Programmer Documentation

ALL-IN-1 Office Menu Installation Guide
ALL-IN-1 Office Menu System Manager's Guide
ALL-IN-1 Office Menu Release Notes
ALL-IN-1 Office Menu Read Me First
ALL-IN-1 Office Menu Application Programmer's Reference
Volume 1
Volume 2
Volume 3
ALL-IN-1 Office Menu Programmer's Mini-Reference

Customization Documentation

ALL-IN-1 Documenter's Kit

ALL-IN-1 Office Menu Style Guide
ALL-IN-1 Office Menu Writer's Guide
Tape of Getting Started Guide and User's Reference

The following documents cover VAX/VMS concepts and VMS layered products that are integrated into or accessible from ALL-IN-1.

Application Control and Management System (ACMS)

VAX ACMS Application Management Guide

DATATRIEVE (DTR)/Database (DBMS)/Common Data Dictionary (CDD)

VAX CDD Utilities Reference Manual
Introduction to VAX DATATRIEVE
VAX DATATRIEVE User's Guide
VAX DATATRIEVE Reference Manual
VAX DATATRIEVE Guide to Using Graphics
VAX DBMS Introduction to Data Manipulation
VAX DBMS Database Administration Reference Manual
VAX DBMS Programming Reference Manual

WPS-PLUS Editor

WPS-PLUS/VMS Getting Started
WPS-PLUS/VMS Editor Functions
WPS-PLUS/VMS List and Sort Processing
WPS-PLUS/VMS Quick Lookup

EDT Editor

VAX EDT Reference Manual

Forms Management System (FMS)

VAX-11 FMS Utilities Manual

Programming Languages

VAX-11 BASIC Language Reference Manual
VAX-11 BLISS-32 Language Reference Manual
VAX-11 COBOL Language Reference Manual
VAX-11 FORTRAN Language Reference Manual
VAX-11 PASCAL Language Reference Manual
VAX-11 PL\1 Language Reference Manual

VMS Operating System

VAX/VMS Document Set
VAX/VMS Release Notes

Conventions

The following conventions are used throughout this document:

Dot Matrix	Indicates prompts and messages from the computer.
lowercase	Lowercase text in an ALL-IN-1 function call, or in a DCL command or command procedure, indicates that you substitute a valid expression in that place.
UPPERCASE	Indicates keyboard keys and Gold functions.
Enter	Indicates information the user puts into a menu or form.
Press	Indicates that the user invokes a single keyboard command (for example, press EXIT SCREEN).
Type	Indicates characters the user puts into a document, program module, or mail message from the keyboard.
Gold X	Indicates that the user presses the Gold key and then the X key to invoke a keyboard command.
CTRL/X	Indicates that the user holds down the CTRL key while pressing X to invoke a keyboard command.

[ ]	Indicates an optional expression in ALL-IN-1 function calls and DCL command strings.
. . .	Indicates that the example contains more data than is shown.
. . .	Indicates that the user can enter additional parameter values or information.
< >	Are used in ALL-IN-1 boiler plate forms to set off areas that contain data to be processed (for example, <OA$TIME>).
|	Indicates alternatives (for example, /INT_CHARACTER={A|B|D} means A, or B, or D).

7

The Subprocess

7.1 Introduction

ALL-IN-1 creates a subprocess (when needed) which you can use to execute DIGITAL Command Language (DCL) command procedures, run programs, and interact directly with VMS through DCL. The subprocess serves as a window to VMS while retaining the context of ALL-IN-1.

In order to communicate with this subprocess, ALL-IN-1 creates three VMS mailboxes and assigns them the logical names SYS$INPUT, OAMAILBOX, and DCLMAILBOX in the subprocess logical name table. The mailboxes are discussed in the following sections.

7.2 Initializing the Subprocess

By default, ALL-IN-1 defers creating the subprocess until it is needed during the session. Specify the /NODEFER_SUBPROCESS command-line qualifier when you want to create the subprocess during initialization. This and other command-line qualifiers are covered in the *System Manager's Guide.*

After the subprocess is created, ALL-IN-1 sends it a log-in command procedure, OAINI.COM, to execute. OAINI.COM typically contains statements that assign (or reassign) process logicals and create (or recreate) DCL symbols.

Although the subprocess is running in the context of ALL-IN-1, it is a new process: all system and group logical-name assignments remain valid, but the subprocess does not recognize symbols and process-level logicals that you created/assigned before running ALL-IN-1. Use OAINI.COM in the subprocess as you would LOGIN.COM to create the symbols and logicals you need in the subprocess.

ALL-IN-1 first searches for OAINI.COM in your default ALL-IN-1 directory, OAUSER. If it does not find it there, it searches the ALL-IN-1 library directory, OA$LIB. ALL-IN-1 also searches OA$DATA. Besides logical-name assignment and symbol creation, OAINI.COM can contain commands that run programs.

See the *System Manager's Guide* for a detailed discussion on log-in files.

7.3 Communicating with the Subprocess Through Mailboxes

VMS mailboxes are record-oriented, input/output devices for general communications between processes. ALL-IN-1 implements one mailbox internally, assigning it the logical name SYS$INPUT and using it to communicate with the subprocess. That is, SYS$INPUT for the subprocess is a mailbox. This allows ALL-IN-1 to execute DCL commands on the user's behalf. The other two mailboxes, OAMAILBOX and DCLMAILBOX, pass records, status information, return codes, messages or any other data between the subprocess and the main process.

The COMMAND and DCL functions, as well as GET OA$DCL, automatically use these mailboxes to communicate with the subprocess. You can call the COMMAND function from the main ALL-IN-1 process to execute command procedures in the subprocess. You can also use the DCL function to run command procedures, but DCL is intended for the execution of single DCL commands, as is the special GET symbol OA$DCL.

Figure 7-1 illustrates the flow control pathways through mailboxes between the subprocess and the main ALL-IN-1 process. The mailboxes are described in the following sections.

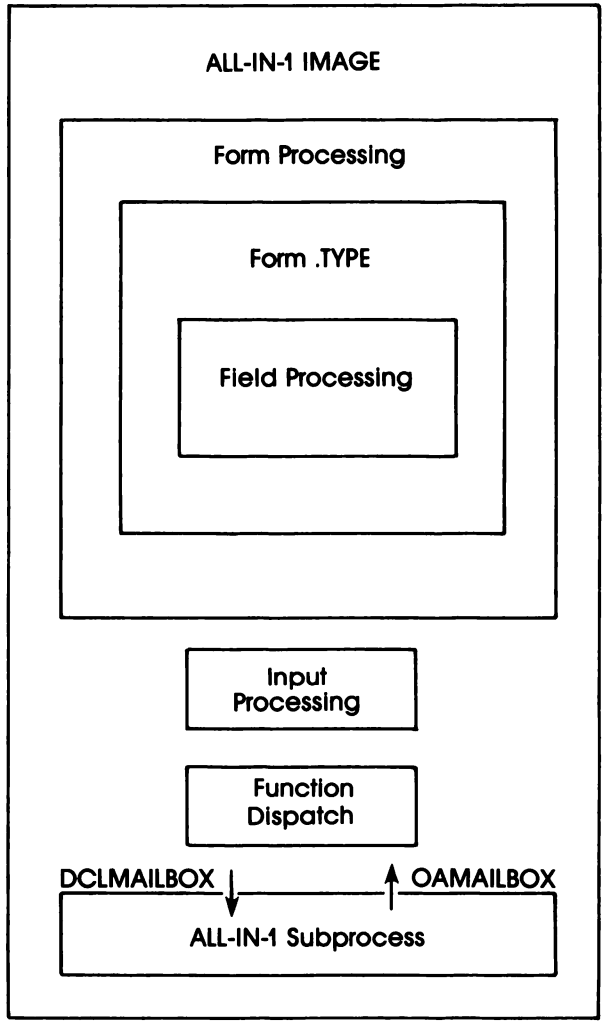


Figure 7-1 Flow Control Between the Subprocess and Main Process

7.3.1 SYS$INPUT

SYS$INPUT is the automatic mailbox through which ALL-IN-1 passes the parameters of ALL-IN-1 functions that use the subprocess. When you enter a $ in a menu CHOICE field, when you press Gold $ in any field, or when you directly call the DCL or COMMAND functions, or GET OA$DCL, you are sending a command through the SYS$INPUT mailbox to the subprocess, where DCL processes it as a standard command.

For example, if the command procedure DEMO.COM consists of the single line: $ SHOW DEFAULT, then the following commands all produce the same response:

(Prompted) User Input	System Response

VMS level (not running ALL-IN-1):

```
$ SHOW DEFAULT
$ @DEMO
```

In the CHOICE field, any menu:

```
$ SHOW DEFAULT
<DCL SHOW DEFAULT                DISK$:[000000]
$ @DEMO

<COMMAND DEMO
```

In any field, after pressing Gold $:

```
$ SHOW DEFAULT
$ @DEMO
```

NOTE: Although these and other commands produce identical results when executed in different ways, some DCL commands do not work this way. The COMMAND and DCL functions in Chapter 6 cover these restrictions.

7.3.2 OAMAILBOX and DCLMAILBOX

Use OAMAILBOX to communicate from the subprocess to ALL-IN-1, and use DCLMAILBOX to communicate from ALL-IN-1 to the subprocess. These mailboxes allow an application running in the subprocess to call ALL-IN-1 to perform functions, and they allow ALL-IN-1 to return the

results of those function calls to the subprocess. Since ALL-IN-1 automatically performs a DCL OPEN on OAMAILBOX when it creates the subprocess, you can use the DCL WRITE command by itself to write messages to OAMAILBOX. The DCL syntax for WRITEs to OAMAILBOX is:

```
$ WRITE OAMAILBOX "OA data"
```

where OA<space> is a required prefix in this context, and the quotes around OA data are required DCL syntax.

When the subprocess is active, that is, when DCL is currently processing a command that has been passed through the SYS$INPUT mailbox, ALL-IN-1 is sensitive to data from the OAMAILBOX. If the subprocess sends back data through this mailbox, ALL-IN-1 reads it and holds it in a buffer. If the subprocess WRITEs additional lines through OAMAILBOX, ALL-IN-1 appends these lines to the buffer. This is useful for long strings of data, because the mailbox record length is limited to 512 bytes.

When the subprocess issues a read to DCLMAILBOX, ALL-IN-1 takes the contents of the buffer built with reads from OAMAILBOX and executes the buffer as a function. After dispatching the function(s), ALL-IN-1 issues any return-status information, followed by an end-of-file [EOF], to DCLMAILBOX. The DCL syntax for reads from DCLMAILBOX is:

```
$ @DCLMAILBOX:
```

If the functions called through OAMAILBOX return any data through DCLMAILBOX, then the subprocess must be prepared to read that return-status information. If the subprocess does not read all records from DCLMAILBOX, a hang condition may result. *To avoid a hang condition, you should always follow each WRITE or series of WRITEs to OAMAILBOX from the subprocess by the @DCLMAILBOX: statement.*

If the subprocess should fail (possibly because it was deleted), OAMAILBOX receives a message indicating this. ALL-IN-1 then issues an error message and aborts the subprocess activity immediately. Flow continues when ALL-IN-1 performs the next function or displays the previous form.

7.4 WRITE OAMAILBOX Syntax

The full syntax for WRITEs to OAMAILBOX depends on the nature of the information passed. Examples are given in this section, in Chapters 6 and 8, and in the VAX/VMS documentation set.

OAMAILBOX is most commonly used in DCL command procedures to call an ALL-IN-1 function or list of functions. For example:

```
$ WRITE OAMAILBOX "OA GET PROFIL.TITLE[OA$USER]"
$ @DCLMAILBOX:
```

ALL-IN-1 dispatches the GET function, and returns the TITLE field from the User Profile record of the current user through the DCLMAILBOX in the DCL symbol RESULT.

To interactively get the title of a specific user, you have to enclose that username in quotes:

```
<GET PROFIL.TITLE["SMITH"]
```

But if you issue this GET through OAMAILBOX, as shown previously, DCL erroneously interprets the first double quotes around SMITH as the end of the WRITE string. Since ALL-IN-1 recognizes the apostrophe as a quote, you can call GET in one of two ways:

```
$ WRITE OAMAILBOX "OA GET PROFIL.TITLE['SMITH']"
$ @DCLMAILBOX:
```

OR

```
$ WRITE OAMAILBOX "OA GET PROFIL.TITLE[""SMITH""]"
$ @DCLMAILBOX:
```

Since DCL interprets double apostrophes as symbol-substitution characters, the following is invalid:

```
$ WRITE OAMAILBOX "OA GET PROFIL.TITLE[''SMITH'']"
$ @DCLMAILBOX:
```

This expression results in the function call:

```
GET PROFIL.TITLE['']
```

since SMITH' is not an existing DCL symbol. You may, however, substitute existing DCL symbols anywhere in a WRITE to OAMAILBOX; for example:

```
$ WRITE OAMAILBOX "OA GET PROFIL.TITLE[""",USER,"""]"
$ @DCLMAILBOX:
```

In this example, DCL scans the WRITE OAMAILBOX command, replaces USER with its DCL-assigned value, and WRITEs the OA GET function call to the OAMAILBOX buffer for the main process to execute. In the main process, ALL-IN-1 interprets the GET and returns the proper TITLE through DCLMAILBOX to the subprocess.

You can specify more than one WRITE to OAMAILBOX before asking for results back from DCLMAILBOX. To continue a single function call over two or more WRITEs, include the continuation mark (-) as the last character in quotes:

```
$ WRITE OAMAILBOX "OA GET 'The time is currently ' OA$TIME -"
$ WRITE OAMAILBOX "OA ', and the date is ' OA$DATE"
$ @DCLMAILBOX:
```

ALL-IN-1 appends the data passed in the second WRITE as is to the data stored from the first WRITE (minus the OA prefix) and dispatches the following function call:

```
GET 'The time is currently ' OA$TIME ', and the date is ' OA$DATE
```

ALL-IN-1 does not display the result of this GET on line 24; rather, it returns the result through DCLMAILBOX.

When you specify multiple WRITEs in sequence without the continuation character, ALL-IN-1 treats each new WRITE as a new function in a list of functions, and adds a backslash to the end of the statement. For example:

```
$ WRITE OAMAILBOX "OA .IF OA$MSG_SEV EQS 'W' OR OA$MSG_SEV EQS 'E' -"
$ WRITE OAMAILBOX "OA THEN .FX GET #KEYWORDS = 'HELP ERRORS ' OA$MSG_ID -"
$ WRITE OAMAILBOX "OA ELSE .FX GET #KEYWORDS = "" """
$ WRITE OAMAILBOX "OA GET OA$FUNCTION = 'HELP ' #KEYWORDS"
$ @DCLMAILBOX:
```

ALL-IN-1 interprets this sequence as two separate functions (or, more correctly, a SCRIPT directive called as a function followed by a regular function) and dispatches the following line:

```
.IF OA$MSG_SEV. . . ELSE .FX GET #KEYWORDS = " "\GET OA$FUNCTION = 'HELP . . .
```

Note that in both cases, OA is required syntax for writing to OAMAILBOX.

Note also that none of the ALL-IN-1 functions are actually performed *until* the DCLMAILBOX mailbox is read with the @DCLMAILBOX: reference. At that time, each specified function is performed in the sequence specified by the ordering of each WRITE OAMAILBOX reference.

More examples follow.

```
$ ! Three functions in one write
$ !
$ WRITE OAMAILBOX "OA COMMAND SUBCOM\CLEAR\WAIT"
$ @DCLMAILBOX:
$ !
$ ! One function in two writes
$ !
$ WRITE OAMAILBOX "OA FORM MAIN MENU/CHOI-"
$ WRITE OAMAILBOX "OA CE=CHOICE/TITLE=TITLE"
$ @DCLMAILBOX:
$ !
$ ! Two functions in two writes
$ !
$ WRITE OAMAILBOX "OA CLEAR"
$ WRITE OAMAILBOX "OA GET OA$FORM_NAME"
$ @DCLMAILBOX:
$ !
$ ! Simple one-form loop
$ !
$ WRITE OAMAILBOX "OA FORM SEARCHFRM\ IFEXIT \ REPEAT"
$ @DCLMAILBOX:
$ !
$ ! Four function calls in a loop
$ !
$ WRITE OAMAILBOX "OA FORM ACCARG/FIELDS=TOTAL"
$ WRITE OAMAILBOX "OA IFEXIT \ FORM AENTER \ REPEAT"
$ @DCLMAILBOX:
```

In the previous example, SEARCHFRM, ACCARG, AND AENTER are not known forms. If you change the names to known forms in ALL-IN-1 and use the /REPEAT qualifier, an infinite loop results. To get out of this infinite loop, use CTRL/C.

7.5 Defining and Using Symbols

Chapter 8 covers the types and uses of ALL-IN-1 symbols. As explained there, an ALL-IN-1 symbol is simply a construct that represents a value. That value may be stored in a symbol table, a data file, a field on the current form, or it may just be a quoted string, used once.

Through the SYS$INPUT and DCLMAILBOX mailboxes, you can send ALL-IN-1 symbol values from the main process to the subprocess. Conversely, you can assign values to ALL-IN-1 symbols through WRITEs from the subprocess to OAMAILBOX. The values you specify in these WRITEs are often stored in DCL symbols created in command procedures.

The following sections describe how symbols are created and how symbol values are sent between the main process and the subprocess.

7.5.1 Creating DCL Symbols with /COPY and /PUT_SAVE

The /COPY field qualifier takes an optional symbol argument. When you specify a permanent symbol here, ALL-IN-1 creates (or updates) a global DCL symbol with the same value and the same name, less the $, as the permanent symbol.

Similarly, the /PUT_SAVE field qualifier creates (or updates) a global DCL symbol for each permanent symbol you specify as a qualifier argument. ALL-IN-1 creates these symbols regardless of how you called the qualified form: that is, if you are not currently in the subprocess, the symbols are still created there.

For example:

Named Data

```
5 Name OLD
/VALID = FORMAT/COPY = DESC,DESC,$DESC_VALUE

6 Name NEW
/COPY = ,FRMAT,$FRMAT_VALUE

7 Name TITLE
/PROMPT = "Enter a Title for this item"/PUT_SAVE = $AAA

8 Name PRIORITY
/PUT_SAVE = $PRIORITY1, #TEMP1, $PRIORITY2
```

If the user exits this form in the usual way the following occurs:

1. /COPY copies the contents of FORMAT.DESC[selected-key] to field OLD and saves the same value in ALL-IN-1 permanent symbol $DESC_VALUE. /COPY also creates global DCL symbol DESC_VALUE (or updates it, if it already exists) for the life of the subprocess.

2 /COPY copies the contents of field NEW on the current form to field FRMAT on the current form and saves the same value in permanent symbol $FRMAT_VALUE. /COPY also creates global DCL symbol FRMAT_VALUE (or updates it, if it already exists) for the life of the subprocess.

3 /PUT_SAVE saves the contents of field TITLE in $AAA. /PUT_SAVE also creates global DCL symbol AAA (or updates it, if it already exists) for the life of the subprocess.

4 /PUT_SAVE saves the contents of PRIORITY in three symbols: local symbol #TEMP1 and permanent symbols $PRIORITY1 and $PRIORITY2. /PUT_SAVE also creates global DCL symbols PRIORITY1 and PRIORITY2 (or updates them, if they already exist) for the life of the subprocess.

7.5.2 Creating DCL Symbols with Functions

Several functions generate global DCL symbols when you call them with a particular combination of parameters. Typically, when a function accepts an optional *output destination* parameter, and that parameter is a permanent symbol, the function creates a global DCL symbol. Functions generate these symbols regardless of how they are called.

The following functions generate DCL symbols.

COMPUTE

When COMPUTE is called with one or more permanent symbols as output destinations, it creates a global DCL symbol for each of those permanent symbols. CREATE gives the DCL symbol the same value and the same name, less the $, as the corresponding permanent symbol.

Examples are given in Chapter 6.

DATE_CONVERT

The first DATE_CONVERT parameter is a symbol that is given a converted date value. When this parameter is a permanent symbol, DATE_CONVERT creates a corresponding global DCL symbol. DATE_CONVERT gives the DCL symbol the same value and the same name, less the $, as the permanent symbol.

Examples are given in Chapter 6.

GET

When you call GET with one or more permanent symbols as output destinations, GET creates a global DCL symbol for each of those permanent symbols. GET gives the DCL symbol the same value and the same name, less the $, as the corresponding permanent symbol.

Examples are given in Chapter 6.

MAKE_FILE_NAME

The first MAKE_FILE_NAME parameter is a symbol that contains the result of the function: an RMS file name. When this parameter is a permanent symbol, MAKE_FILE_NAME creates a corresponding global DCL symbol. MAKE_FILE_NAME gives the DCL symbol the same value and the same name, less the $, as the permanent symbol.

Examples are given in Chapter 6.

NEXT_LIST
OA$FLD_NEXT_LIST

NEXT_LIST and OA$FLD_NEXT_LIST return a record from a sequential file and load the value of that record in permanent symbol $NEXT_LIST_ITEM and in global DCL symbol NEXT_LIST_ITEM.

An example is given in Chapter 6.

PARSE_USER

The PARSE_USER function creates a valid VMS file-name root from a supplied parameter and loads that value in permanent symbol $PARSED_USER_NAME and in global DCL symbol PARSED_USER_NAME.

SAVE

The first SAVE parameter is a symbol that is assigned a literal-string value. When this parameter is a permanent symbol, SAVE creates a corresponding global DCL symbol. SAVE gives the DCL symbol the same value and the same name, less the $, as the permanent symbol.

An example is given in Chapter 6.

7.6 Common DCL Symbols

When you call the functions covered in the preceding section in a given manner, they assign values to specific DCL symbols. These same functions, as well as others, can also return values to the subprocess through common DCL symbols.

When you call a function from the subprocess and the call does not specify (or does not allow) an output destination parameter but still generates a tangible value, then ALL-IN-1 stores that value in either (or both) of the common DCL symbols. These symbols, RESULT and TERMINATOR, are covered in the following paragraphs.

RESULT

When you call a function from the subprocess that generates a value, ALL-IN-1 may return that value through DCLMAILBOX and store it in RESULT. This feature allows command procedures or programs running in the subprocess to obtain information from the main process. RESULT can be treated as any other global DCL symbol in the command procedure.

Application programs that use OAMAILBOX and DCLMAILBOX in the subprocess can also access the contents of RESULT.

When you call the following functions in the following ways from the subprocess, they return a value in RESULT:

COMPUTE

When you call COMPUTE from the subprocess with no output destination, it returns the value of the specified expression in RESULT.

An example is given in Chapter 6.

GET

When you call GET from the subprocess with no output destination, it returns the value of the symbol list in RESULT.

An example is given in Chapter 6.

LOGICAL

When you call LOGICAL with one parameter, a logical name from the subprocess, it returns the value of the equivalent name in RESULT.

PROMPT

When you call PROMPT from the subprocess, ALL-IN-1 returns any user response that precedes the terminator key or key sequence in RESULT. The terminator value is saved in special symbol TERMINATOR.

PROMPT_ID

When you call PROMPT_ID from the subprocess, ALL-IN-1 returns any user response that precedes the terminator key or key sequence in RESULT. The terminator value is saved in special symbol TERMINATOR.

WAIT

When WAIT is called from the subprocess, any user response that precedes the terminator key is returned in RESULT. The terminator value is saved in special symbol TERMINATOR.

YESNO_PROMPT

When you call YESNO_PROMPT from the subprocess, the user's response — either yes, no, y, or n — that precedes the terminator key or key sequence is returned in RESULT. The terminator value is saved in special symbol TERMINATOR.

YESNO_PROMPT_ID

When you call YESNO_PROMPT_ID from the subprocess, the user's response — either yes, no, y, or n — that precedes the terminator key or key sequence is returned in RESULT. The terminator value is saved in special symbol TERMINATOR.

TERMINATOR

Several ALL-IN-1 functions are interactive and depend on user input to complete tasks. Often, this user input takes the form of a single keystroke: the WAIT function, for example, causes ALL-IN-1 to pause until the user presses a valid terminator such as RETURN, <ESC>, and so on.

The interactive functions are FORM, FIELD, PROMPT, PROMPT_ID, WAIT, YESNO_PROMPT, and YESNO_PROMPT_ID. The PROMPT, PROMPT_ID, WAIT, YESNO_PROMPT and YESNO_PROMPT_ID functions assign a value to common DCL symbol TERMINATOR that corresponds to the terminator key or key sequence that the user pressed to

end the function. ALL-IN-1 stores the same numeric value in special symbols OA$FORM_TERMINATOR and OA$FIELD_TERM, but ALL-IN-1 creates and updates TERMINATOR directly in the subprocess. It is sometimes more efficient to read this symbol if you have called one of these interactive functions through OAMAILBOX.

By testing TERMINATOR, you can determine what key a user pressed in response to a PROMPT, PROMPT_ID, WAIT, YESNO_PROMPT or YESNO_PROMPT_ID; and knowing how that key is defined (EXIT SCREEN, RETURN, and so forth), you can decide what to do next. For example:

```
$ !
$ ! See if the user wants to continue...
$ !
$ WRITE OAMAILBOX "OA PROMPT"
$ @DCLMAILBOX:
$ !
$ ! Test for an EXIT SCREEN (TERMINATOR value of 112):
$ !
$ IF TERMINATOR .EQ. 112 THEN GOTO EXIT_SCREEN
$ !
$ ! Else, the user wants to continue...
$ !
```

TERMINATOR values are listed in Table 4-4.

NOTE: In earlier versions, ALL-IN-1 used the TERMINATOR symbol to determine user intentions. However, some of the functions (FORM, FIELD) that used to assign values to TERMINATOR no longer do, and the functions that do set TERMINATOR in ALL-IN-1 Version 2.X assign it a value in a different way.

ALL-IN-1 now has internal terminator functions that assign values to a special symbol, OA$FORM DISPOSE. There is also a special function, IFEXIT, that automatically tests for EXIT SCREEN types of responses.

NOTE: Each VAX installation usually has locally defined symbols and commands that access installation applications. You should check with the System Manager for details on what symbols are automatically defined for the user at log-in time. If you need any of these symbols within the subprocess, you must also define them within all users' OAINI.COM file because system-wide symbols are not automatically defined in the ALL-IN-1 subprocess.

Since you can call several functions through OAMAILBOX before asking for results from DCLMAILBOX, it is very possible that two or more of those functions attempt to return data in a common DCL symbol, either RESULT and TERMINATOR. For example:

```
$ ! Get the terminal type, store it in DCL symbol TERMINAL.
$ ! Get the equivalent name of logical OA$LIB at the same time.
$ !
$ WRITE OAMAILBOX "OA GET OA$TERMINAL\LOGICAL OA$LIB"
$ @DCLMAILBOX:
$ TERMINAL := 'RESULT
```

Contrary to the comments in this command procedure, local DCL symbol TERMINAL contains the equivalent name, if any, of logical OA$LIB. Both GET and LOGICAL are assigning values to RESULT, and since the LOGICAL function is called last, the equivalent name of OA$LIB is stored in RESULT (and thus in TERMINAL).

The following command procedure also fails to work as stated:

```
$ ! Get the value of logical OAHELP, display it on line 24.
$ !
$ WRITE OAMAILBOX "OA LOGICAL OAHELP\DISPLAY ''RESULT'"
$ @DCLMAILBOX:
```

In this case, common symbol RESULT is assumed to contain the equivalent name of logical OAHELP. However, ALL-IN-1 does not update RESULT with the expected value until the command procedure reads DCLMAILBOX.

When calling functions that accept output-destination parameters, you can avoid these types of conflicts by specifying permanent symbols as output destinations. For example:

```
$ ! Get the terminal type, store it in permanent symbol $TERM.
$ ! Get the equivalent name of logical OA$LIB at the same time.
$ !
$ WRITE OAMAILBOX "OA GET $TERM=OA$TERMINAL\LOGICAL OA$LIB"
$ @DCLMAILBOX:
$ WRITE SYS$OUTPUT "You are working with a ''TERM', and the current"
$ WRITE SYS$OUTPUT "ALL-IN-1 library directory is ''RESULT'"
```

7.6.1 Passing Command-Procedure Parameters to the Subprocess from ALL-IN-1

You can pass constant values in the form of DCL command-file parameters to the subprocess through COMMAND function calls. The syntax is:

```
COMMAND com-file [/DCL_qual=arg] [P1 ... P8]
```

where P1 through P8 are positional DCL parameters. Using DCL parameters allows you to develop generic command procedures that perform the same basic operations when executed, with slight differences based on the parameter(s) passed.

For example, the following COMMAND function calls from Named Data involve three menu options but only one command procedure:

Named Data

```
1 Name A
COMMAND UPDATE ADD

2 Name C
COMMAND UPDATE CHG

3 Name D
COMMAND UPDATE DEL
```

In this example, command procedure UPDATE.COM can test parameter P1 and perform one of three different operations depending on the parameter passed: ADD, CHG, or DEL. This is a DCL feature available whenever command procedures are executed and wherever ALL-IN-1 COMMAND function calls are recognized. The VAX/VMS documentation set discusses parameters in more detail.

7.7 ALL-IN-1 Logicals

In addition to the mailbox names, ALL-IN-1 creates other logical names during initialization as process-level logicals in the subprocess. You modify these logicals and create new ones with the DCL DEFINE and ASSIGN commands.

To call those ALL-IN-1 functions and field qualifiers that accept logical names as a valid parameter or argument, the logical name must be defined in the subprocess. The functions which recognize logical names must accept a file specification as a parameter of that function. Some examples are:

Functions

COMMAND
DCL
DELETE_FILE
DO
EDIT
LIST
MAKE_FILE_NAME
MERGE
PURGE_FILE
QUEUE_BATCH
QUEUE_PRINT
SCRIPT
SORT_LIST
TEST_SPEC

Field Qualifiers

/LIST
/SCROLL

However, ALL-IN-1 also creates three of the system-assigned subprocess logicals — OAUSER, OADATA, and OALIB — in the main process for the duration of the ALL-IN-1 session. If these logicals are defined in the main process logical-name table, you can reassign them with the LOGICAL function. Because LOGICAL does not affect logicals created in the subprocess, it is possible to have one value for OALIB in the main process, where it is recognized and translated during initialization; and another value in the subprocess, where it is recognized and translated like any other subprocess logical. This is described in Table 7-1.

Table 7-1 ALL-IN-1 Subprocess Logicals

Logical Name	Equivalent Name	Description
DCLMAILBOX	"MBAxxx:"	VMS mailbox from main process to subprocess.
OAUSER*	"dev:[dir]"	User's default ALL-IN-1 directory. Equivalent at initialization to PROFIL.DIRECT[OA$USER].
OADATA* **	"dev:[dir]"	Location of shared data files.
OALIB* **	"dev:[dir]"	Location of ALL-IN-1 image and other library files.
OA$LOGGER	< >	VMS mailbox for ALL-IN-1 Logging facility.
OAMAILBOX	"__MBAyyy:"	VMS mailbox from subprocess to main process.
SYS$COMMAND	"__MBAzzz:" OR "__TTmn:"	Source of input for the VMS command.
SYS$DISK	"dev:[dir]"	System disk.
SYS$ERROR	"__TTmn:"	Destination for system error messages: usually SYS$OUTPUT.
SYS$INPUT	"__MBAzzz:" OR "__TTmn:"	Source of input for this image or procedure.
SYS$LOGIN	"dev:[dir]"	User's log-in directory.
SYS$OUTPUT	"____TTmn:"	Destination for output for this process.
TT	__MBAzzz:	The SYS$INPUT mailbox.

*OAUSER, OADATA, and OALIB are also created in the main process.
**The OA$DATA and OA$LIB logicals are not created if they have already been assigned values in one of the logical name tables (system, group, or process).

7.8 Running Command Procedures and Programs

Command procedures and most programs that you run stand-alone on VMS can be interfaced to ALL-IN-1 through the subprocess.

NOTE: You can also run command procedures and programs in a spawned subprocess: that is, one which you create with the SPAWN function. However, a subprocess created with SPAWN has none of the features of the primary subprocess that is covered in this chapter.

SPAWN is useful for running applications or portions of applications that may not need to communicate with the main ALL-IN-1 process or with another application running in the subprocess.

7.8.1 Running Command Procedures

7.8.1.1 The COMMAND Function. The COMMAND function executes DIGITAL Command Language (DCL) command procedures in the subprocess. COMMAND function calls are valid wherever function calls are recognized, except from the subprocess.

The syntax for COMMAND calls is:

```
COMMAND com-file [/OUTPUT = file_spec] [P1 ... P8]
```

The VAX/VMS documentation set gives detailed information on command procedures.

7.8.1.2 The DCL Function. The DCL function allows you to enter the subprocess and use it interactively. As with the COMMAND function, DCL is valid wherever function calls are recognized, except from inside the subprocess.

You can call DCL interactively by entering a $ in a menu choice field. Although you can invoke command procedures with this function, the COMMAND function is more versatile.

7.8.1.3 GET OA$DCL. GET OA$DCL also allows you to enter the subprocess and use it interactively. To do this, enter $ < GET OA$DCL = "command" in a menu choice field.

7.8.2 Obtaining User Input in a Program or Command Procedure

Standard applications and subsystems that execute some of their tasks in the subprocess generally call ALL-IN-1 forms through OAMAILBOX to gather data. The data is generally returned by ALL-IN-1 in the form of global DCL symbols. While this form-based interface is fine for many applications, it may be inappropriate for others.

Consider the following example of an interactive program, VAXMail (the VMS MAIL utility), running in the subprocess. Since incomplete DCL commands are invalid in command procedures, a procedure containing the single line:

```
$ MAIL
```

causes errors. If users are familiar with VAXMail, they can enter the command at a **$** prompt and answer the resulting MAIL> prompts:

```
$ MAIL
MAIL>SEND MEMO.TXT
TO: MORRISON
SUBJECT: NATAL DAY
MAIL>EXIT
```

However, you can give VAXMail, or any similar program, an ALL-IN-1 form-based interface that gathers the same sort information in a way that is consistent with existing ALL-IN-1 subsystems. You can then pass the data to VAXMail in one of several ways.

For example, VAXMail has a SEND operation that you can invoke at DCL level. Define an argument form, MSGSEND, with three fields

corresponding to the addressee, the subject, and the file to send. Create DCL symbols from those arguments with the /PUT_SAVE qualifier:

Named Data

```
1 Name .TYPE
ARG

2 Name TO
/PUT_SAVE = TO

3 Name SUBJEC
/PUT = SUBJECT

4 Name FILE
/PUT = FILE
```

The command procedure to implement this SEND would contain the following lines:

```
$ WRITE OAMAILBOX "OA FORM MSGSND\GET OA$FORM_DISPOSE"
$ @DCLMAILBOX:
$ IF "'RESULT'" .EQS. "2" THEN $MAIL/SUBJECT = "'SUBJECT'" 'FILE' 'TO'
```

Another way to run VAXMail from a command procedure is to pass it parameters as data lines. You can send data lines to programs to override prompting (for those programs that prompt for user input by default). These data lines satisfy DCL requests for input, and unwanted messages are bypassed. For example:

```
$ ! Invoke VAXMail W/ RUN command to allow recognition of the data
$ ! lines that follow: the SEND command, the recipient, the subject,
$ ! and the text of the message.
$ !
$ RUN SYS$SYSTEM:MAIL
SEND MEMO.TXT
MORRISON
Meeting on the 24th
$ EXIT
```

There is one drawback to sending parameters as data lines: DCL does not substitute symbols in a data line. For example, this procedure sends file MEMO.TXT to MORRISON every time you run it.

To overcome this limitation, create a temporary command procedure with the DCL OPEN command and build it using the WRITE command. Symbol substitution is then possible:

```
$ ! Display the form, get the parameters
$ !
$ WRITE OAMAILBOX "OA FORM MSGSND\GET OA$FORM_DISPOSE"
$ @DCLMAILBOX:
$ !
$ ! Open a temporary file
$ !
$ OPEN/WRITE TMP MAIL.TMP
$ WRITE TMP "$ MAIL"
$ WRITE TMP "SEND ''FILE'"
$ WRITE TMP"''TO'"
$ WRITE TMP "''SUBJECT'"
$ !
$ ! Close the temporary file
$ !
$ CLOSE TMP
$ !
$ ! Execute the temporary file
$ !
$ @MAIL.TMP
$ !
$ ! Delete the temporary file
$ !
$ DELETE MAIL.TMP;0
```

You can also obtain user input by redirecting terminal input. When the user needs to interact directly with a VMS program or utility in the subprocess, redirect the SYS$INPUT stream to the user's terminal with the ASSIGN statement. Typical programs which need to do this are text editors and DATATRIEVE. The command is:

```
$ ASSIGN/USER 'F$LOGICAL("SYS$OUTPUT") ' SYS$INPUT
```

NOTE: Specify SYS$OUTPUT in uppercase letters.

This ASSIGN command determines which physical terminal the output stream is assigned to and then assigns that terminal to the input stream.

The /USER qualifier only makes the assignment for the duration of the next image that executes in the command procedure. For example:

```
$ ! Redirect the terminal input
$ !
$ ASSIGN/USER 'F$LOGICAL("SYS$OUTPUT") ' SYS$INPUT
$ !
$ ! Initialize the editor with commands from file COMFILE.EDT and
$ ! edit the file named by the symbol "EDITFILE".
$ !
$ EDIT/EDT/COMMAND=COMFILE.EDT 'EDITFILE'
```

8

ALL-IN-1 Symbols

8.1 Introduction

In the DIGITAL Command Language, symbols are variables. You create a DCL symbol by equating it with an initial value in an assignment statement. In ALL-IN-1, symbol refers not only to VMS-type symbols but also to anything that references a data set. Thus, an ALL-IN-1 symbol can be a quoted literal string that references itself, an FMS field name that references the data stored in that corresponding field of a data file, a special symbol that references DATATRIEVE data, or a DCL-type symbol stored in a symbol table. This chapter covers what ALL-IN-1 symbols look like, how to create them, how to reference and combine them, and where to use them.

8.2 Basic Symbol Types

8.2.1 Quoted String

ALL-IN-1 treats quoted strings as symbols, and in many instances, you can use quoted strings interchangeably with other types of symbols. Quote refers to both quotation marks and apostrophes, both of which mean the same thing to ALL-IN-1. If a quoted string is to contain an embedded quote, you can represent this as follows:

- Use two quotes to represent the embedded quote. For example:

  'The System Manager''s account'

- Use a different quote character for the embedded quote. For example:

  "The System Manager's account"

8.2.2 Field Name

You can reference the FMS-defined names of fields on the current or most recently processed form as symbols, with two restrictions: the field name cannot contain a dollar sign ($) or a period (.). The following field names are valid symbol references:

USER
USER_2
ADDRESS

ALL-IN-1 limits field names to 30 characters in length.

8.2.3 Permanent Symbol Table

The permanent symbol table is an indexed data file that ALL-IN-1 creates as OAUSER:parsed-username.PST and saves from one session to the next. This file is indexed by the names of permanent symbols; the remainder of a record is simply the value of the symbol. Permanent symbols are global to ALL-IN-1, and you can create and update them at will.

Preface permanent symbols with a dollar sign ($). The following are valid permanent symbol names:

$USERNAME
$$SPECIAL
$USER_NAME

Permanent symbol names are limited to 30 characters in length and must contain only letters, numbers, $, or an underscore (_). Do not include embedded spaces in the symbol name.

8.2.4 Special Symbols

ALL-IN-1 special symbols are predefined values that allow access to:

- Variable system information such as current time and VMS process statistics
- Internal ALL-IN-1 variables such as the terminal type
- Context information such as the current form and current editor style
- Miscellaneous information such as DATATRIEVE variables

Special symbols are prefaced with a two or three character facility code such as DTR or OA, followed by a $.

Most special symbols are system-maintained and read-only, but you can update several of them. The following are examples of special symbols:

OA$USER
DTR$ADDRESS
STAT$PID

The facility code helps to identify the type of information that is stored in the symbol. For example, a symbol prefixed with DTR contains a DATATRIEVE-generated value, and a STAT$ symbol contains process statistics.

8.2.5 Local Symbol Table

The local symbol table is a high-speed temporary symbol table maintained internally for the duration of an ALL-IN-1 session. You can use local symbols to pass arguments and other data between ALL-IN-1 facilities during the session; ALL-IN-1 deletes the table when you exit. Preface local symbols with a pound sign (#). The following are valid local symbol names:

#F1
#CAL_USER
#WORK_A

ALL-IN-1 limits local symbol names to 30 characters in length.

8.2.6 Data Set References

Data set references (DSRs) are multipart, read-only symbols that refer to the contents of one or more fields in a specific record of a specific data set. DSRs are self contained: the reference implicitly opens, reads, and closes the data set.

A data set is just that: a set of data. An entry form and its related indexed file is a data set in which the FMS fields on the form generally correspond one-to-one to RMS fields in the file. The File Cabinet is also a data set, one that comprises several data files. Most of these file-cabinet data files are specially formatted and are not accessible through entry forms, so the syntax for file-cabinet DSRs is different than that for entry-form related DSRs.

The syntax of a DSR that refers to one complete field in an entry-form data set is:

```
entry_form.field[key]
```

where

entry_form	is the name of an entry form that contains a .FILE directive in Named Data pointing to an indexed data file.
field	is the name of any field on the form.
key	is the value of the key for the record being accessed.

The period (.) between entry_form and field is required, as are the brackets enclosing the key. DSRs can also specify substrings, alternate keys, and multiple fields. These are described in the following sections.

Appendix C covers data sets in more detail.

8.3 Using Data Set References

The simple data file NAMES.DAT, which corresponds to entry form NAMES, is illustrated below.

key field

fields:

NAME	ADDRESS	PHONE
SMITH	123 Elm St.	230-1234
TAYLOR	456 Maple St.	430-4444
YOUNG	12-B 4th Ave.	221-2211

To return the address of TAYLOR, specify the DSR:

```
NAMES.ADDRESS["TAYLOR"]
```

To return the phone number of YOUNG, specify:

```
NAMES.PHONE["YOUNG"]
```

The key can also be another field or a symbol-table symbol. For example:

```
NAMES.ADDRESS[USER]
```

where USER is a field on the current form that contains a valid key

OR

```
NAMES.ADDRESS[OA$USER]
```

where OA$USER is a special symbol that contains a valid key.

8.3.1 Combining and Nesting DSRs

Although the key values given in the preceding DSR examples are literals, symbols, and field names, you can specify any symbol combination that translates into a valid key, even another DSR. For example, the following DSR is valid:

```
EFORM.FIELDA[ NEWDB.FIELDB[OA$CURDOC] ]
```

Specifying this DSR returns the value of FIELDA from entry form EFORM. This particular value is indexed by FIELDB of form NEWDB, as indexed in turn by special symbol OA$CURDOC.

To return several fields within the same record, supply multiple field names. For example, to obtain the ADDRESS and PHONE of TAYLOR, specify:

```
NAMES.ADDRESS["TAYLOR"] .PHONE
```

This DSR returns the concatenated string: 12-B 4th Ave.221-2211. Because the data-set name and key are omitted for the .PHONE reference, ALL-IN-1 assumes that .PHONE refers to the last referenced data set. Another way to return the same information is to specify:

```
NAMES.["TAYLOR"] .ADDRESS .PHONE
```

To format the fields for printing, add spaces between each reference by inserting a quoted space between the fields:

```
NAMES.["TAYLOR"] .ADDRESS " " .PHONE
```

OR

```
NAMES.["TAYLOR"] .ADDRESS ' ' .PHONE
```

8.3.2 Specifying Alternate Keys with DSRs

Any RMS key in a file can be specified as an alternate key by specifying the key name and its value as:

```
file_form:alt-key.field[key-value]
```

Alt-key is the RMS name of a field in the file and key-value is the value of the key field. The colon (:) is required syntax and is distinct from the colons used to indicate the length and position of a symbol substring. Alternate key values do not have to be unique.

The following example returns the NAME or NAMEs that correspond to the address enclosed in quotes:

```
<GET NAMES:ADDRESS.NAME["123 Elm St."]
```

8.3.3 Special Field Names

You can use several special field names in DSRs to provide additional capabilities:

- %ADDRESS — The %ADDRESS special field name returns the relative file address (RFA) of a field in a file. The RFA is the RMS zero-offset starting point of the field and can be used as a key value in subsequent DSRs.

  .%ADDRESS is useful when retrieving key values from files that allow duplicate primary keys. For example:

  ```
  <GET #ORIG = NAMES.%ADDRESS["WADE"]
  ```

  Local symbol #ORIG now contains the RFA of the first occurrence of user Wade in NAMES.DAT. If you add other Wades later ALL-IN-1 stores them, alphabetially, in the file before this Wade. However, this and similar GETs always refers to the first occurrence of the name by default. You could always reference the original Wade directly by specifying:

  ```
  <GET = NAMES.NAME[#ORIG]
  ```

- %COUNT — To return the number of occurrences of the key value in the file, use the special field name %COUNT. %COUNT is also valid for alternate keys. For example:

  ```
  NAMES.%COUNT["SMITH"]
  ```

  returns the number of entries indexed by the name SMITH in file NAMES.DAT,

  ```
  NAMES:PHONE.%COUNT["221-2211"]
  ```

  returns the number of entries in which the PHONE number field contains the value 221-2211, assuming the phone number field is a RMS key called PHONE.

- %KEY — To return the value in the key field when you do not know the name of the field itself, use the special field name %KEY. For example, if you knew that the key field in file NAMES.DAT also contained the valid key to the file associated with form set PROFIL, but you did not know the name of that field, specifying:

  ```
  PROFIL.EDITOR[NAMES.%KEY[OA$USER]]
  ```

  returns, for the current user (OA$USER), the value of the EDITOR field from the PROFIL file, using the key field from NAMES.DAT.

- %NEXT — The %NEXT special field name returns the key field value for the next record in the file.

  If you have not established a context for the next record, %NEXT returns the first record in the file.

  For example, to establish a context for %NEXT, you could specify:

  ```
  <GET NAMES:ADDRESS.ADDRESS["SMITH"]
  ```

  ALL-IN-1 returns the address 456 Maple St. If you then specify:

  ```
  <GET NAMES:ADDRESS.%NEXT[""]
  ```

  ALL-IN-1 returns the address of user TAYLOR; repeating this call again returns the address of user YOUNG, and so on.

  .%NEXT is similar to, but faster than, the NEXT_LIST function and selection lists.

- %WHOLE — To return an entire record as a string rather than individual fields within the record, use the special field name %WHOLE. For example, to get the entire YOUNG record from NAMES.DAT, specify:

  ```
  NAMES.%WHOLE["YOUNG"]
  ```

Figure 8-1 summarizes how entry-form related DSRs obtain data.

FORMAT.DESC["RUNOFF"] ①

FORMAT ②

```
Document Handling (Format) Master

Name:              ③          (field FORMAT)
Description:       ④          (field DESC)
Default filename:
Object extension:
DSAB type:
Template filename:

Edit function:
                              Initialization File:
                              Purge Count:
                              Editor Type:

List function:
Print function:
Standard formatting?          Y

Compile Command:
Template Form:
```

1 Name .TYPE
ENTRY /MODE = UPDATE/OVERLAY/hard = 'Document Format Master'

2 Name .FILE ⑤
OA$DATA:FORMAT.DAT,FORMAT

3 Name FORMAT
/RECOGNITION = FORMAT:FORMAT/SHOW = DESC/hard = 'Format Master Name'

4 Name DESC
/hard = 'Description:'
:
:

⑤ OA$DATA:FORMAT.DAT

Key field

FORMAT ③	DESC ④	other fields....
RUNOFF	RUNOFF text formatter	
....		

FORMAT.DESC["RUNOFF"] ① refers to the FORMAT data set which contains the file OA$DATA:FORMAT.DAT ⑤ . The entry form FORMAT ② describes this file. This DSR extracts the value of field DESC ④ from the record in the file whose key ③ is the literal string RUNOFF. The DSR returns the value: RUNOFF text formatter.

Figure 8-1 Obtaining Data with a DSR

8.4 Symbol Substrings

Nonliteral symbols can specify a substring of the symbol value. The syntax for specifying substrings is consistent across all types of symbols:

```
              field:len:pos
  special$symbol:len:pos
         $symbol:len:pos
         #symbol:len:pos
data_set."literal":len:pos[key]
```

where :len is the length of the substring and :pos is the zero-origin position of the substring. The colons are required. When :len is specified, the returned substring is exactly len characters long. If needed it is truncated or padded with blanks. This feature is useful in formatting screen and file output.

Various combinations are possible:

- Omit both :len and :pos to return the entire symbol value.
- Omit the :pos to return the first :len characters of the symbol or to pad the symbol with trailing blanks. For example, if the value of permanent symbol $EDITOC were MEETING MEMO, specify:

  ```
  $EDITDOC:7
  ```

  to return MEETING. To return a 15-character string regardless of the length of $EDITDOC, specify:

  ```
  $EDITDOC:15
  ```

  This would return "MEETING MEMO ".
- Include both :len and :pos to extract an internal substring. For example, if the current system date, OA$DATE_FULL, were formatted as 12-JAN-1983, specifying

  ```
  OA$DATE_FULL:3:3
  ```

  returns JAN. As described above, specifying a value that is too large for :len causes the substring to be padded with trailing spaces. For example, specifying:

  ```
  OA$DATE_FULL:20:3
  ```

  returns "JAN-1983 ".

- Substrings can be extracted with DSRs in the same manner. Using the file NAME.DAT as example, specifying:

  ```
  NAMES:ADDRESS.PHONE:3:0["123 Elm St."]
  ```

  returns the phone exchange(s), (for example, 230), for the individual or individuals who live at 123 Elm St.

You can also specify Z as the :pos argument to zero-suppress a numeral. For example:

```
<GET NUM = "000123" \GET NUM::Z
```

returns " 123" on line 24.

8.5 Symbol Lists

When a function or a field qualifier takes any ALL-IN-1 symbol as a parameter or argument, multiple symbols can be specified to create a single, concatenated value. That is, if you string symbols together, they are concatenated in the same way that fields in a DSR are concatenated.

For example, if multiple symbols are separated by a space in a GET function call, the contents of each symbol are concatenated to form a single return value. Spaces are added between the symbol values only when explicitly specified. For example, the function call:

```
GET OA$DATE OA$TIME:5
```

returns 12-JAN-198308:10, while the function call:

```
GET OA$DATE " " OA$TIME:5
```

returns 12-JAN-1983 08:10.

8.6 Iterative Symbol Substitution

If a symbol contains another symbol name as its value, you can retrieve the value of the inner symbol by preceding the outer symbol with a @. This operator is similar to the DCL ' substitution operator. For example:

<GET #WPDOC = "A document name"	Local symbol #WPDOC contains the quoted string.
<GET #A = "#WPDOC"	Local symbol #A also contains a quoted string, which happens to be the name of another symbol.
<GET #B = @#A	Local symbol #B now contains "A document name", the value stored in #WPDOC.

Iterative substitution can also be done on the left side of an assignment expression (that is, GET destinations, COMPUTE destinations, and so on). Continuing with the preceding example:

<GET @#A = "DOCNAM"	#WPDOC, not #A, contains the value "DOCNAM"

The @ can also be specified more than once to achieve multiple iterative substitution. For example:

<GET @@#A = "A new value"	Field DOCNAM contains the quoted string "A new value".

Symbol substring expressions are ignored during iterative substitutions and are only evaluated after all indirect references have been resolved. For example:

<GET #INNERSYM = "Ultimate value"	#INNERSYM contains the quoted string.
<GET #OUTERSYM = "',' #INNERSYM"	#OUTERSYM contains the quoted string.
<GET #OVERSYM = @#OUTERSYM:15	#OVERSYM contains the following string: ',' #INNERSYM:15
<GET @#OVERSYM	Displays on line 24: ",Ultimate value"

The FOR function defers iterative substitution until it reads a record. Since FOR performs the DO subfunctions after each record is processed, the contents of an inner symbol, and thus the RSE selection criteria, can change during a search. This also allows you to include concatenated symbols in the RSE; for example:

```
FOR PROFIL WITH .USER EQL @"'LIST' #NUMBER" DO . . .
```

If #NUMBER contains the value 01 before this function call, all records where the .USER field matches the expression LIST01 are selected. If the DO function(s) change the value of #NUMBER while processing LIST01, the new value of the RSE is used in subsequent searches.

8.7 Specifying Date/Time Values

Several functions, including CALENDAR, DATE_CONVERT, MAIL, QUEUE_BATCH, and QUEUE_PRINT, take date- and time-related parameters or have qualifiers that take date- and time-related arguments. You can specify dates and times in several ways, as follows:

Dates:

dd-mmm[-yyyy]*
dd-mmm[-yy]*
mm/dd/yyyy
mm/dd/yy
EOM (end of current month)
TODAY
TOMORROW
YESTERDAY

* If you omit the year, the current year is assumed.

Times:

[hh][:mm][:ss][.tt]
[hh][:mm][:ss][.tt][AM|PM]**

** For am/pm format, the defaults if AM or PM are omitted are:

00:00-11:59 = AM
12:00-23:59 = PM

While in AM/PM Mode, times between 12:00 - 23:59 and 00:00-11:59 are also honored.

If 00:00-11:59 is specified, it is taken as 12:00AM-11:59AM.
If 12:00-23:59 is specified, it is taken as 12:00PM-11:59PM.

Offsets:

+[n]D forward n days
-[n]D back n days
+[n]W forward n weeks
-[n]W back n weeks
+[n]M forward n months
-[n]M back n months
+[n]Y forward n years
-[n]Y back n years

With offsets, rounding might occur. For example, moving forward one month on January 31st yields Feb 28th (or 29th if leap year). Moving forward 1 year on Feb 29 yields Feb 28 (although moving forward 4 years on Feb 29 yields Feb 29).

8.7.1 Creating and Updating Symbols and Data Sets

To create or update a symbol interactively from a field on a form, use the /PUT_SAVE and /COPY field qualifiers. /PUT_SAVE puts the final contents of the qualified field into an existing symbol-table symbol or field on the current form, or creates the symbol automatically if it does not exist (assuming it is a symbol-table symbol). Also /PUT_SAVE creates a global DCL symbol with a name that corresponds to the specified symbol-table or field destination.

/COPY also creates permanent or local symbols when specified with an optional third argument.

Several ALL-IN-1 functions create or update permanent or local symbols. The GET and SAVE functions are the most direct ways to create or update a symbol-table symbol. GET recognizes symbol-table symbols, FMS fields, and several special expressions as output-destination parameters and assigns a specified value to that output destination. SAVE, which is used primarily from command procedures, assigns a literal value to an existing permanent or local symbol or creates the symbol automatically if it does not exist.

8.7.2 Getting Values from Symbols

The GET function is the most direct way to retrieve information from symbols.

In addition, the field qualifier /GET_SAVE extracts the value of any symbol and displays it in the qualified field. The common form qualifier /GET is similar to the /GET_SAVE field qualifier: use it to specify values that display in chosen fields when a form is preprocessed.

8.7.2.1 In the Main Process. In the main process, use symbols as parameters in function calls and as arguments for field qualifiers.

8.7.2.2 Values Passed to the Subprocess. Several field qualifiers and functions automatically create DCL symbols with names that correspond to ALL-IN-1 symbol names.

8.7.2.3 Values Passed from the Subprocess. Chapter 7 covers symbol creation in the subprocess and how those symbol values are passed to the main process.

8.7.3 Values Stored in Multiple Symbols

ALL-IN-1 often stores values in more than one type of symbol. The ALL-IN-1 username, for example, is stored in the PROFIL data set as the key field and also in special and permanent symbols. *When a value is stored in both a data set and a symbol-table symbol, ALL-IN-1 always returns the value faster when you retrieve it as a symbol-table symbol.* Although DSRs are more versatile, they must open a data set, read one or more values, and then cache the data set before returning the requested value. In applications where performance is a factor, use symbol-table symbols, preferably local symbols, to store values.

8.8 Special Symbols

ALL-IN-1 special symbols include read-only symbols, write-only symbols and read-write symbols. Table 8-1 lists the special symbols with the following notation:

- The write-only symbols are designated by a *W after their name.
- The read-write symbols are designated by a *RW after their name.
- The read-only symbols have no notation following their name.

Table 8-1 Special Symbols

Symbol	Key	Description
E		Numeric constant; equivalent to 2.71828182845904552353602874713 52
OA$BRDCST_TRAP		Has value 1 if broadcast-message trapping is enabled, 0 if not
OA$CAB_FILE_PURGE_COUNT	*RW	Contains number of file versions to keep after File Cabinet reorganization
OA$CHOICE		Most recent contents of last or current menu CHOICE field; equivalent to field name CHOICE when used as a symbol
OA$CURDOC		FOLDER/DOCNUM of current File Cabinet document; equivalent to CAB$.FOLDER:30[OA$CURDOC] + CAB$.DOCNUM[OA$CURDOC]
OA$CURDOC_AUTHOR		Author of current File Cabinet document; equivalent to CAB$.AUTHOR[OA$CURDOC]
OA$CURDOC_CREATED		Creation date of current document; equivalent to CAB$.CREATED[OA$CURDOC]
OA$CURDOC_CREATED_NBS		Creation date of current document; equivalent to CAB$.CREATEDNBS[OA$CURDOC]
OA$CURDOC_CREDAT		Creation date of current File Cabinet document; equivalent to CAB$.CREDAT[OA$CURDOC]
OA$CURDOC_DAPOINTER		Is P for private or S for shared document; equivalent to CAB$.DAPOINTER[OA$CURDOC]
OA$CURDOC_DELETE		Is-doc-deletable flag; equivalent to CAB$.DELETE[OA$CURDOC]
OA$CURDOC_DOCNUM		Number of current File Cabinet document; equivalent to CAB$.DOCNUM[OA$CURDOC]
OA$CURDOC_DSAB		Internal handling code for current File Cabinet document; equivalent to CAB$.DSAB[OA$CURDOC]

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$CURDOC_FILENAME		VMS file name in which current File Cabinet document is stored; equivalent to CAB$.FILENAME[OA$CURDOC]
OA$CURDOC_FIXER		"FIXER", either DOCNUM or its inverse; equivalent to CAB$.FIXER[OA$CURDOC]
OA$CURDOC_FOLDER		Folder of the current File Cabinet document; equivalent to CAB$.FOLDER[OA$CURDOC]
OA$CURDOC_FORMAT		Format (handling) of current File Cabinet document; equivalent to CAB$.FORMAT[OA$CURDOC]
OA$CURDOC_KEYWORDS		Keywords associated with current File Cabinet document; equivalent to CAB$.KEYWORDS[OA$CURDOC]
OA$CURDOC_MAIL_ORIG		Number of previous mail message; equivalent to CAB$.MAIL_ORIG[OA$CURDOC]
OA$CURDOC_MAIL_STATUS		Status of current mail message (READ, SENT, and so on); equivalent to CAB$.MAIL_STATUS[OA$CURDOC]
OA$CURDOC_MODDAT		Date current File Cabinet document was last modified; equivalent to CAB$.MODDAT[OA$CURDOC]
OA$CURDOC_MODIFIED		Modification date of the current File Cabinet document; equivalent to CAB$.MODIFIED[OA$CURDOC]
OA$CURDOC_MODIFIED_NBS		Modification date of the current File Cabinet document; equivalent to CAB$.MODIFIEDNBS[OA$CURDOC]
OA$CURDOC_MODIFY		Is-doc-modifiable flag; equivalent to CAB$.MODIFY[OA$CURDOC]
OA$CURDOC_ORIGINAL		Name of previous File Cabinet document; equivalent to CAB$.ORIGINAL[OA$CURDOC]

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$CURDOC_REFNUM		Reference number; inverse of DOCNUM; equivalent to CAB$.REFNUM[OA$CURDOC]
OA$CURDOC_SETUP		Name of setup associated with current File Cabinet document (Version 1.X feature); equivalent to CAB$.SETUP[OA$CURDOC]
OA$CURDOC_TITLE		Title of current File Cabinet document; equivalent to CAB$.TITLE[OA$CURDOC]
OA$CURDOC_TYPE		Class of current File Cabinet document; equivalent to CAB$.TYPE[OA$CURDOC]
OA$CURDOC_V1TYPE		Contains the Version 1.X type field
OA$CURMES	*RW	FOLDER/DOCNUM of current mail message; equivalent to CAB$.FOLDER:30[OA$CURMES] + CAB$.DOCNUM[OA$CURMES]
OA$CURMES_ADDRESSEE		First addressee of current mail message; equivalent to CAB$.TO[OA$CURMES]
OA$CURMES_DATE		Date that current mail message was created; equivalent to CAB$.CREATED[OA$CURMES]
OA$CURMES_DOCNUM		Number of current mail message; equivalent to CAB$.DOCNUM[OA$CURMES]
OA$CURMES_FILE		VMS file name of the current mail message; equivalent to CAB$.FILENAME[OA$CURMES]
OA$CURMES_FOLDER		Folder of current mail message; equivalent to CAB$.FOLDER[OA$CURMES]
OA$CURMES_FROM		Sender of current mail message; equivalent to CAB$.FROM[OA$CURMES]

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$CURMES__ORIG__FILE		File name of original mail message
OA$CURMES__ORIG__FROM		Sender of the previous mail message
OA$CURMES__ORIG__SUBJECT		Subject of previous mail message
OA$CURMES__RECEIPT		Value of return-receipt flag; equivalent to CAB$.RR[OA$CURMES]
OA$CURMES__STATUS		Status of current mail message; equivalent to CAB$.MAILSTATUS[OA$CURMES]
OA$CURMES__SUBJECT		Subject of current mail message; equivalent to CAB$.SUBJECT[OA$CURMES]
OA$CURMES__TITLE		Title of current mail message; equivalent to CAB$.TITLE[OA$CURMES]
OA$CURMES__TYPE		Type of current mail message; equivalent to CAB$.TYPE[OA$CURMES]
OA$DATA		Translation of logical name OA$DATA
OA$DATE		Current system date in mm/dd/yy format
OA$DATE__FULL		Current system date in dd-mmm-yyyy format
OA$DATE__NBS		Current system date, NBS format: yyymmddhhmmss
OA$DAY		Day of week (unabbreviated)
OA$DCL	*W	Specifies a command line to be executed by DCL in the subprocess
OA$DEFAULT__DSAB		DSAB type used for default editor
OA$DEFAULT__EDITOR		Default editor type PROFIL.EDITOR(OA$USER)
OA$DEFAULT__EXTENSION		File-type extension corresponding to default edit style (OA$DEFAULT__EDITOR)

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$DEFAULT_MENU		Name of the default menu form
OA$DISPLAY	*W	Specifies text to be displayed on line 24 of the screen
OA$DIST_FOLDER		Name of folder containing distribution lists
OA$DIST_TYPE		Document type of distribution lists
OA$DTR	*W	Specifies a command line to be executed by DATATRIEVE
OA$DTR_AUX_STATUS_LINE		Contents of the auxiliary DATATRIEVE message buffer when DATATRIEVE is invoked from ALL-IN-1
OA$DTR_LAST_LINE		Last line that DATATRIEVE sends to the DATATRIEVE interface
OA$DTR_STATUS		Value returned from last DATATRIEVE call
OA$DTR_STATUS_LINE		Contents of the primary DATATRIEVE message buffer when DATATRIEVE is invoked from ALL-IN-1
OA$DTR_VERSION		Version identification of DATATRIEVE image
OA$EDIT_BUFFER		Line of text to write to the editor buffer
OA$EDITING		Contains the value 1 if the user is in an editor
OA$EDIT_FILE	*RW	VMS name of file to edit when editor is called without that parameter
OA$EDIT_INI_FILE	*RW	VMS name of edit initialization file to use if editor is called without that parameter
OA$EDIT_PARAM	*RW	Value of editor-specific parameter to pass if editor is called without that parameter

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$EDIT_PURGE_COUNT	*RW	Number of copies to keep after editing if editor is called without that parameter
OA$ENTRY_FORM_MODE		Contains one of the following values: 1=Add, 2=Change, 3=Delete, 4=Inquire, 5=Copy, 6=Update, 7=Entry
OA$FIELD_NAME		FMS name of current field
OA$FIELD_TERM		Numeric value of key pressed to exit current field; equivalent to OA$FORM_TERMINATOR
OA$FIELD_TEXT		Contents of current field; equivalent to OA$FIELD_NAME
OA$FORM_DISPOSE		Numeric disposition of current field: 0 – EXIT, 1 – NEXT, 2 – DONE, 3 – STAY
OA$FORM_NAME		Current form name
OA$FORM_TERMINATOR		Numeric value of key pressed to exit current form; equivalent to OA$FIELD_TERM
OA$FULL_NAME		User's full name; equivalent to PROFIL.FULNAM[OA$USER] or $FULLNAME
OA$FUNCTION	*W	Specifies a line to be sent to the ALL-IN-1 command dispatcher
OA$HARDCOPY		Hard Copy Mode flag: 1 = yes, 0 = no
OA$LIB		Translation of logical name OA$LIB
OA$LOG	*W	Specifies a line to be sent to the log file
OA$MAIL_COUNT		Indicates the number of UNREAD messages
OA$MAIL_COUNT_DISPLAY		A text string indicating the number of UNREAD messages
OA$MAIL_READ		Read status string; equivalent to default language word for READ

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$MAIL_SENT		Sent status string; equivalent to default language word for SENT
OA$MAIL_TYPE		Mail type string; equivalent to default language word for TYPE
OA$MAIL_UNREAD		Unread status string; equivalent to default language word for UNREAD
OA$MAIL_UNSENT		Unsent status string; equivalent to default language word for UNSENT
OA$MAIL_UNSENT_COUNT		String indicating number of UNSENT messages
OA$MAIL_VERSION		Version of mail
OA$MEETING_COUNT_DISPLAY		Number of meetings scheduled that are not answered
OA$MENU_CHOICE		Most recent contents of current menu choice field; field name CHOICE when used as a symbol
OA$MENU_COMMAND		Current menu command
OA$MENU_FORM		Name of the current menu form
OA$MENU_LEVEL		Menu stacking level
OA$MENU_PREVIOUS	*RW	Name of the previous menu form
OA$MENU_REMAINDER	*RW	Remainder of stacked menu options
OA$MERGE_KEY		Current item from MERGE list file; equivalent to OA$MERGE_REC
OA$MERGE_LINE	*RW	Outputs text to the MERGE destination
OA$MERGE_REC		Current item from MERGE list file; equivalent to OA$MERGE_KEY
OA$MESSAGE_PAUSE	*W	Number of seconds for the message to pause on the screen; valid in Script Mode
OA$MESSAGE_SUPPRESS	*W	Flag: 0 = display messages; 1 = do not display messages

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$MR_ACTIVE		Flag: 0 = the Message Router is active; 1 = the Message Router is not active
OA$MSG_FAC		Facility prefix of most recent primary message (usually OA)
OA$MSG_ID		Message identification of the most recent primary message
OA$MSG_SEV		Severity code of most recent primary message: I — Informational, S — Success, W — Warning, E — Error, F — Fatal
OA$MSG_TEXT		Text of most recent primary message
OA$NO		Default language word for "no"
OA$NODE		Name of current VAX node
OA$PAGE_LENGTH	*RW	Current page length of MERGE output file
OA$PAGE_NUMBER	*RW	Number of current page in MERGE output file
OA$PORT	*W	Sends a line of text to the printer port; you must previously have used PORT_ON
OA$PREV_FIELD		Previous field referenced; valid only in Help context
OA$PREV_FORM		Previous form referenced; valid only in Help context
OA$PREV_HFIELD		Field name specified by the /HARD qualifier; valid only after you have stacked forms
OA$PREV_HFORM		Form name specified by the /HARD qualifier; valid only after you have stacked forms
OA$PRIMARY_NODE		In a cluster, the name of a node in the cluster that is known to all other Message Routers. Use in preference to OA$NODE
OA$PRIV_DCL		Specifies that the user has access to DCL
OA$PROCESS_NAME		Main Process

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$PRODUCT_NAME		Product name of the ALL-IN-1 image
OA$PROFIL_ACTITEM		Flag: 1 = ALL-IN-1 notifies the user of any pending action items when the user first runs ALL-IN-1; 0 = ALL-IN-1 does not notify the user; equivalent to PROFIL.ACTITEM[OA$USER]
OA$PROFIL_ADDR1		Address line 1 in the profile record for the user's address; equivalent to PROFIL.ADDR1[OA$USER]
OA$PROFIL_ADDR2		Address line 2 in the profile record for the user's address; equivalent to PROFIL.ADDR2[OA$USER]
OA$PROFIL_ADDR3		Address line 3 in the profile record for the user's address; equivalent to PROFIL.ADDR3[OA$USER]
OA$PROFIL_ADDR4		Address line 4 in the profile record for the user's address; equivalent to PROFIL.ADDR4[OA$USER]
OA$PROFIL_CALDAY:		Line number on which the currently selected day displays in the Advanced Calendar option; equivalent to PROFIL.CALDAY[OA$USER]
OA$PROFIL_CALTIMEINC		Smallest time increment (1 minute in format HH::MM) for which the Calendar facility schedules meetings and appointments; equivalent to PROFIL.CALTIMEINC[OA$USER]
OA$PROFIL_CLASS		Contents of a user-definable field used by the Logging facility; equivalent to PROFIL.CLASS[OA$USER]
OA$PROFIL_CORP_PHONE		Privilege of user to add entries to or edit the corporate phone directory; equivalent to PROFIL.CORP_PHONE[OA$USER]
OA$PROFIL_DEPART		User's department, division, unit and so forth (24 characters); equivalent to PROFIL.DEPART[OA$USER]

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$PROFIL_DIRECT		User's default ALL-IN-1 directory; equivalent to PROFIL.DIRECT[OA$USER]
OA$PROFIL_EDITOR		User's default editor; equivalent to PROFIL.EDITOR[OA$USER]
OA$PROFIL_ENDH		Time the user's business day ends; equivalent to PROFIL.ENDH[OA$USER]
OA$PROFIL_FRMLIB		Any user-defined form libraries; equivalent to PROFIL.FRMLIB[OA$USER]
OA$PROFIL_FULNAM		32 character symbol with the user's full name in a format chosen by the user; equivalent to PROFIL.FULNAM[OA$USER]
OA$PROFIL_LANGUAGE		Language of the user; equivalent to PROFIL.LANGUAGE[OA$USER]
OA$PROFIL_MAIDES		Default mail destination for the user (for example, ALL-IN-1, VAXMAIL or PRINTER); equivalent to PROFIL.MAIDES[OA$USER]
OA$PROFIL_MAIL_MENU		Default mail style: EMC = ALL-IN-1 style mail or EMD = DECMAIL style mail; equivalent to PROFIL.MAIL_MENU[OA$USER]
OA$PROFIL_MAIL_READ_RECEIPT		Notification to sender when user has read mail; equivalent to PROFIL.MAIL_READ_ RECEIPT[OA$USER]
OA$PROFIL_MEALE		Time at which the user's lunch period ends; equivalent to PROFIL.MEALE[OA$USER]
OA$PROFIL_MEALS		Time at which the user's lunch period starts; equivalent to PROFIL.MEALS[OA$USER]

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$PROFIL_MULTI_NODE		User has an entry in a multinode user profile; equivalent to PROFIL.MULTI_NODE[OA$USER]
OA$PROFIL_NODAYS		Number of days (1-7) in the user's work week; equivalent to PROFIL.NODAYS[OA$USER]
OA$PROFIL_NODE		Node at which the user is working; equivalent to PROFIL.NODE[OA$USER]
OA$PROFIL_NOTIFY		Notification on arrival of new mail; equivalent to PROFIL.NOTIFY[OA$USER]
OA$PROFIL_OPER		Not currently used but reserved by DIGITAL for future use; equivalent to PROFIL.OPER[OA$USER]
OA$PROFIL_PHONE		User's phone number; equivalent to PROFIL.PHONE[OA$USER]
OA$PROFIL_PRINT		User's default print destination; equivalent to PROFIL.PRINT[OA$USER]
OA$PROFIL_PRINTER_PORT		Availability of a printer port on the user's terminal; equivalent to PROFIL.PRINTER_PORT[OA$USER]
OA$PROFIL_PRVCMD		Privilege to enter ALL-IN-1 functions interactively; equivalent to PROFIL.PRVCMD[OA$USER]
OA$PROFIL_PRVDCL		Privilege to access DCL directly from ALL-IN-1 and create a subprocess; equivalent to PROFIL.PRVDCL[OA$USER]
OA$PROFIL_PRVERR		Enabling of error logging for the user; equivalent to PROFIL.PRVERR[OA$USER]
OA$PROFIL_PRVLOG		Enabling of statistical logging for the user; equivalent to PROFIL.PRVLOG[OA$USER]

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$PROFIL_PRVSRC		Enabling of full field recognition (Gold S); equivalent to PROFIL.PRVSRC[OA$USER]
OA$PROFIL_PRVSUP		Enabling of user access to supervisor-only fields on forms; equivalent to PROFIL.PRVSUP[OA$USER]
OA$PROFIL_QBNOT		Batch queue notification; equivalent to PROFIL.QBNOT[OA$USER]
OA$PROFIL_QPNOT		Print queue notification; equivalent to PROFIL.QPNOT[OA$USER]
OA$PROFIL_SET_MODE		Default mode in which the user's terminal operates (Screen or Hard Copy Mode); equivalent to PROFIL.SET_MODE[OA$USER]
OA$PROFIL_START		Default form which displays when the user first enters ALL-IN-1; equivalent to PROFIL.START[OA$USER]
OA$PROFIL_STARTD		Day on which the user's work week starts; equivalent to PROFIL.STARTD[OA$USER]
OA$PROFIL_STARTH		Hour at which the user's work day starts; equivalent to PROFIL.STARTH[OA$USER]
OA$PROFIL_STATUS		Contents of a user-definable field that may contain status information such as: VACATION, SICK, ON CALL and so on; equivalent to PROFIL.STATUS[OA$USER]
OA$PROFIL_TICKLER		Notifies the users of pending reminders when the user first enters ALL-IN-1; equivalent to PROFIL.TICKLER[OA$USER]
OA$PROFIL_TIMEFMT		Format in which time is specified (24 hour or AM/PM format); equivalent to PROFIL.TIMEFMT[OA$USER]

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$PROFIL_TITLE		30 character symbol containing the user's job title in a user preferred format; equivalent to PROFIL.TITLE[OA$USER]
OA$PROFIL_UFLAG1		User-definable flag containing a 1 bit value; equivalent to PROFIL.UFLAG1[OA$USER]
OA$PROFIL_UFLAG2		User-definable flag containing a 1 bit value; equivalent to PROFIL.UFLAG2[OA$USER]
OA$PROFIL_UFLAG3		User-definable flag containing a 1 bit value; equivalent to PROFIL.UFLAG3[OA$USER]
OA$PROFIL_UFLAG4		User-definable flag containing a 1 bit value; equivalent to PROFIL.UFLAG4[OA$USER]
OA$PROFIL_UFLAG5		User-definable flag containing a 1 bit value; equivalent to PROFIL.UFLAG5[OA$USER]
OA$PROFIL_UFLAG6		User-definable flag containing a 1 bit value; equivalent to PROFIL.UFLAG6[OA$USER]
OA$PROFIL_UFLAG7		User-definable flag containing a 1 bit value; equivalent to PROFIL.UFLAG7[OA$USER]
OA$PROFIL_UFLAG8		User-definable flag containing a 1 bit value; equivalent to PROFIL.UFLAG8[OA$USER]
OA$PROFIL_UFLAG9		User-definable flag containing a 1 bit value; equivalent to PROFIL.UFLAG9[OA$USER]
OA$PROFIL_UFLAG10		User-definable flag containing a 1 bit value; equivalent to PROFIL.UFLAG10[OA$USER]

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$PROFIL_USER		30 character symbol containing the ALL-IN-1 username for the current user; equivalent to PROFIL.USER[OA$USER]
OA$PROFIL_VMSUSR		30 character symbol containing the user's VMS username; equivalent to PROFIL.VMSUSR[OA$USER]
OA$PROFIL_ZIPCOD		User's zipcode; equivalent to PROFIL.ZIPCOD[OA$USER]
OA$PROMPT_DISPOSE		Numeric disposition of current prompt
OA$PROMPT_HELP	*RW	Keyword list to pass to HELP when HELP is called from a prompt
OA$PROMPT_ID		While the prompt is being executed and the PROMPT_ID or YESNO_PROMPT_ID function is used, the value is the prompt's ID, else, the value is "*"; if you are not on a prompt the symbol is empty
OA$PROMPT_TEXT		User response entered at prompt; equivalent to RESULT when PROMPT is called from the subprocess
OA$SCRIPT_EXIT_HANDLER	*RW	Function to serve as a Script exit handler
OA$SCRIPT_PSIB	*RW	Script pseudo-input buffer
OA$SCRIPT_TERM		Terminator of pseudo-input buffer
OA$SCRIPT_TEXT		Text of pseudo-input buffer minus the terminator
OA$SCROLL_FIELD		Name of the last field which has a /SCROLL qualifier
OA$SEL_ADDRESS		Relative file address of a record selected during FOR function call; equivalent to %ADDRESS special field name
OA$SEL_COUNT		Total number of records selected during a FOR function call

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$SEL_DSAB		Name of file containing FOR-selected records
OA$SEL_KEY		Value of current key selected by FOR function
OA$SEL_LINE		Line number of current key selected during FOR call
OA$SRC_COUNT		Number of records searched in a FOR function call or a SEARCH form
OA$STATUS	*RW	Status of last function call
OA$SUBSYSTEM	*RW	Name of current ALL-IN-1 subsystem
OA$TERM_DEVICE		VMS device name of SYS$OUTPUT
OA$TERMINAL		Specific terminal type (see Appendix C)
OA$TERMINAL_7BIT_CSI		Specifies that the terminal responds with a CSI (Character Sequence Introducer) and is set to /NOEIGHTBIT
OA$TERMINAL_ANSICRT		Specifies that the terminal conforms to ANSI CRT programming standards
OA$TERMINAL_APP_KEYPAD		Specifies that the keypad is set to Application Keypad Mode to enable the DEFINE/KEY facility
OA$TERMINAL_AUTOBAUD		Disposition of automatic baud rate detection
OA$TERMINAL_AVO		Specifies the value of the AVO (Advanced Video Option) bit
OA$TERMINAL_BLOCK		Specifies the ability to operate the terminal in Block Mode Transmission
OA$TERMINAL_BRDCSTMBX		Specifies whether or not broadcast messages are sent to an associated mailbox
OA$TERMINAL_CRFILL		Specifies the need to generate fill characters following the pressing of RETURN

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$TERMINAL_DCL_CTRLC		Specifies whether DCL has set up an associated CTRL/C trap for this process
OA$TERMINAL_DCL_MAILBX		Specifies whether there is an associated mailbox for a spawned process
OA$TERMINAL_DCL_OUTBND		Specifies whether out-of-bound ASTs are to be delivered to the parent of a spawned process
OA$TERMINAL_DECCRT		Specifies whether or not the terminal conforms to DIGITAL VT100-family standards
OA$TERMINAL_DECCRT2		Specifies that the terminal supports DECCRT2 capabilities (VT200-family terminals)
OA$TERMINAL_DIALUP		Specifies that the terminal is a dial-up terminal
OA$TERMINAL_DISCONNECT		Specifies that the process connected to this terminal may or may not be discontinued if the line detects a hang-up
OA$TERMINAL_DRCS		Specifies that the terminal supports Dynamically Redefinable Character Sets (loadable character fonts)
OA$TERMINAL_EDIT		Specifies whether the terminal is capable of performing ANSI defined advanced editing functions
OA$TERMINAL_EDITING		Specifies that line editing is allowed
OA$TERMINAL_EIGHTBIT		Specifies that the terminal uses an eight-bit ASCII character code
OA$TERMINAL_ESCAPE		Specifies whether the ANSI standard escape sequences transmitted from the terminal are handled as a single multiple-character terminator
OA$TERMINAL_FALLBACK		Specifies that eight-bit DIGITAL Multinational Character Set characters display on the terminal in seven-bit representation

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$TERMINAL_FMS_SUPP		Specifies that the terminal supports FMS
OA$TERMINAL_HALFDUP		Specifies if the terminal's mode of operation is full-duplex or half-duplex
OA$TERMINAL_HANGUP		Specifies whether or not the terminal is hung-up when you log out
OA$TERMINAL_HOLDSCREEN		Specifies whether or not the operation of the Scroll key is enabled or disabled on VT52 and VT55 terminals
OA$TERMINAL_HOSTSYNC		Specifies whether the terminal can synchronize the flow of input from the terminal by the generation of X/OFF and X/ON
OA$TERMINAL_INSERT		Specifies that the terminal allows you to insert characters when you are editing command lines
OA$TERMINAL_LFFILL		Specifies whether the system must generate fill characters following a line feed on the terminal
OA$TERMINAL_LK201		Specifies that the terminal has a LK201 keyboard
OA$TERMINAL_LOCALECHO		Specifies whether the terminal echoes characters locally rather than relying on the host to echo them
OA$TERMINAL_LOWER		Indicates whether or not lowercase characters are available
OA$TERMINAL_MBXDSABL		Indicates that mailboxes are disabled
OA$TERMINAL_MECHFORM		Specifies that the terminal supports mechanical form feeds
OA$TERMINAL_MECHTAB		Specifies that the terminal supports mechanical tabs
OA$TERMINAL_MODEM		Indicates if the terminal is connected to a modem or to a cable that supplies standard EIA modem control signals

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$TERMINAL_MODHANGUP		Specifies that terminal hang-up can be modified without privilege
OA$TERMINAL_NOBRDCST		Specifies that ALL-IN-1 traps all broadcast messages
OA$TERMINAL_NOECHO		Specifies that the terminal does not echo characters locally
OA$TERMINAL_NOSCROLL		Specifies that the terminal does not support scrolling
OA$TERMINAL_NOTYPEAHD		Specifies that the terminal only accepts input when a program, or the system, issues a read to it
OA$TERMINAL_PAGE		Specifies the page length of the terminal's output
OA$TERMINAL_PASSALL		Specifies that all input and output data is in 7- or 8-bit binary format (no data interpretation occurs)
OA$TERMINAL_PASTHRU		Specifies that all input and output data is in 7- or 8-bit binary format (no data interpretation occurs)
OA$TERMINAL_PORT		Specifies that a terminal has a printer port
OA$TERMINAL_READSYNC		Specifies whether the system solicits read data from a terminal using CTRL/S and terminates the read using CTRL/Q
OA$TERMINAL_REGIS		Specifies whether the terminal understands REGIS graphic commands
OA$TERMINAL_REMOTE		Specifies that the terminal is a dial-up terminal
OA$TERMINAL_SCOPE		Specifies that the terminal is a video terminal
OA$TERMINAL_SECURE		Specifies whether the BREAK key on the terminal logs out the current process or if none is present, initiates the log-in sequence

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$TERMINAL_SIXEL		Specifies that the terminal supports SIXEL Graphics Instruction Sets
OA$TERMINAL_SYSPWD		Specifies that LOGIN requires the system password before the user name prompt displays
OA$TERMINAL_TTSYNC		Specifies whether the terminal synchronizes output by responding to CTRL/S and CTRL/Q
OA$TERMINAL_TYPE		Specifies the terminal type
OA$TERMINAL_TYPE_A1		Specifies the terminal type used by ALL-IN-1
OA$TERMINAL_UDK		Specifies that the terminal supports User Defined Keys
OA$TERMINAL_WIDTH		Specifies the number of characters on each input or output line
OA$TERMINAL_WRAP		Specifies that the terminal generates a carriage return when it reaches the end of a line
OA$TIME		Current system time in hh:mm format
OA$TITLE		User's title; equivalent to PROFIL.TITLE[OA$USER]
OA$TM_DATE		Current calendar date
OA$TM_DAY		Day of week on which OA$TM_DATE occurs
OA$TM_DAY1		Day of week of first graph line
OA$TM_DAY2		Day of week of second graph line
OA$TM_DAY3		Day of week of third graph line
OA$TM_DAY4		Day of week of fourth graph line
OA$TM_DAY5		Day of week of fifth graph line
OA$TM_DAY6		Day of week of sixth graph line

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$TM_DAY7		Day of week of seventh graph line
OA$TM_GRAPH1		First line of time of day graph
OA$TM_GRAPH2		Second line of time of day graph
OA$TM_GRAPH3		Third line of time of day graph
OA$TM_GRAPH4		Fourth line of time of day graph
OA$TM_GRAPH5		Fifth line of time of day graph
OA$TM_GRAPH6		Sixth line of time of day graph
OA$TM_GRAPH7		Seventh line of time of day graph
OA$TM_GRAPH_TITLE		Time of day graph header line (time bar)
OA$TM_MONTH		Name of month in a small month calendar (menus)
OA$TM_MONTH1		First week of small month calendar
OA$TM_MONTH2		Second week of small month calendar
OA$TM_MONTH3		Third week of small month calendar
OA$TM_MONTH4		Fourth week of small month calendar
OA$TM_MONTH5		Fifth week of small month calendar
OA$TM_MONTH6		Sixth week of small month calendar
OA$TM_MONTH_DISPLAY		Name of month for a display month
OA$TM_OWNER		User name of current calendar owner
OA$TM_WEEK1		First week in a display month
OA$TM_WEEK2		Second week in a display month
OA$TM_WEEK3		Third week in a display month
OA$TM_WEEK4		Fourth week in a display month
OA$TM_WEEK5		Fifth week in a display month
OA$TM_WEEK6		Sixth week in a display month
OA$TM_YEAR		Year for small month display

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$TXL_SEARCH_LAST	*W	Enables/disables the alternate search order for TXL elements
OA$TM_YEAR_DISPLAY		Year for display month
OA$USER		Username; equivalent to $USERNAME
OA$VERSION		Current ALL-IN-1 version and base-level
OA$WASTEBASKET		File-cabinet wastebasket folder name
OA$YES		Default language word for YES
PI		Numeric constant; equivalent to 3.14159265358979323846264338327
STAT$ASTCNT		Process AST count
STAT$ASTLM		Process AST limit
STAT$BIOCNT		Buffered I/O count
STAT$BIOLM		Buffered I/O limit
STAT$BUFIO		Count of processed buffered I/O operations
STAT$BYTCNT		Byte count
STAT$BYTLM		Byte limit
STAT$CPULIM		CPU time limit
STAT$CPUTIM		CPU time used
STAT$DFWSCNT		The default workspace size
STAT$DIOCNT		DIO remaining count
STAT$DIOLM		DIO limit
STAT$DIRIO		Count of I/O operations performed
STAT$FILCNT		Fill character count
STAT$FILLM		Fill character limit
STAT$GPGCNT		Global page count
STAT$JOBCNT		Job count

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
STAT$LOGINTIM		Log-in time
STAT$PAGFLTS		Page faults
STAT$PID		Process ID
STAT$PPGCNT		Page count
STAT$PRCCNT		Count of active processes
STAT$PRI		Priority of the process
STAT$PRIB		Base priority of the process
STAT$TQCNT		Timer queue count
STAT$TQLM		Timer queue limit
STAT$UIC		Process UIC
OA$WPL_GET_AP		Autopagination flag; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_BM		Bottom margin value; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_CM		Column margin; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_CP		Copies count; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_DA		Normal/dark flag; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_DD		Document destination; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_EX		Extra 1/2 line spacing; only meaningful when WPS-PLUS is available under ALL-IN-1

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$WPL_GET_FR		Starting page to print; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_IP		Initial page number; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_LO		Language of origin; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_PI		Character pitch; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_PM		Left margin; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_PS		Page size; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_R1		Replacement character 1; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_R2		Replacement character 2; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_RSNUM		Restore print settings; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_SE		Sheet feeder setting; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_SP		Stop at every page; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_SSNUM		Same print settings number; only meaningful when WPS-PLUS is available under ALL-IN-1

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$WPL_GET_TM		Top margin; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_TO		Last page to print; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_VP		Vertical pitch; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_WEF_AD		Set ABBRV document; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_WEF_CT		Current text size; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_WEF_LD		Set LIBRY document; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_WEF_MA		Math Y/N flag; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_GET_WEF_ST		Standard text size; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_AP	*W	Autopagination flag; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_BM	*W	Bottom margin value; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_CM	*W	Column margin; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_CP	*W	Copies count; only meaningful when WPS-PLUS is available under ALL-IN-1

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$WPL_PUT_DA	*W	Normal/dark flag; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_DD	*W	Document destination; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_EX	*W	Extra 1/2 line spacing; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_FR	*W	Starting page to print; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_IP	*W	Initial page number; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_LO	*W	Language of origin; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_PI	*W	Character pitch; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_PM	*W	Left margin; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_PS	*W	Page size; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_R1	*W	Replacement character 1; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_R2	*W	Replacement character 2; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_RSNUM	*W	Restore print settings; only meaningful when WPS-PLUS is available under ALL-IN-1

(continued)

Table 8-1 Special Symbols (Cont.)

Symbol	Key	Description
OA$WPL_PUT_SE	*W	Sheet feeder setting; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_SP	*W	Stop at every page; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_SSNUM	*W	Same print settings number; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_TM	*W	Top margin; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_TO	*W	Last page to print; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_VP	*W	Vertical pitch; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_WEF_AD	*W	Set ABBRV document; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_WEF_CT	*W	Current text size; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_WEF_LD	*W	Set LIBRY document; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_WEF_MA	*W	Math Y/N flag; only meaningful when WPS-PLUS is available under ALL-IN-1
OA$WPL_PUT_WEF_ST	*W	Standard text size; only meaningful when WPS-PLUS is available under ALL-IN-1

8.9 Permanent Symbols

Permanent symbols are maintained for each ALL-IN-1 account in username.PST, where username is the ALL-IN-1 username. ALL-IN-1 subsystems create and maintain a number of permanent symbols which you can update. You can create additional permanent symbols with GET and other functions.

The permanent symbols are listed in Table 8-2.

Table 8-2 Permanent Symbols

Symbol	Description
$ACTNAM	Name of an action item
$AI_CLASS	Stores the class of last entered action item
$AI_CLASS_SAVE	Current class (first selection field)
$AI_DATE_SAVE	Current items date
$AI_ITEM	Stores the item code for the last entered action item
$AI_ITEM_SAVE	Current item (second selection field)
$AP_DATE_SAVE	Date of current appointment
$AP_TIME_SAVE	Time of current appointment (first selection field)
$A_PHNUMBER	Stores current ALL-IN-1 user phone number
$A_PHONE_NUMBER	Symbol for ALL-IN-1 directory phone number
$A_PHONE_USER	Name of current selected ALL-IN-1 directory entry
$A_PH_PER_NUM	Personal phone for current ALL-IN-1 directory selection
$BURST_PAGE	Burst page? Y/N
$CAL_REM_NUMBER	Number of next reminder to be created

(continued)

Table 8-2 Permanent Symbols (Cont.)

Symbol	Description
$CAL_TODO_NUMBER	Number of next to-do list item
$CNV_FILE	File to convert
$CURUDP	Name of the current UDP
$C_PHNUMBER	Stores current corporate phone number
$C_PHONE_USER	Name of current selected corporate directory entry
$C_PH_PER_NUM	Personal phone for current corporate directory selection
$DICTIONARY	Stores current dictionary name for use by CDD
$DISTFOLDER	Folder name of distribution list
$DISTNAME	Title of distribution list
$DOMAIN	Stores current domain for use by DATATRIEVE
$DTDOC	Current document transfer document
$EDITDOC	Name of the current document for editing operations
$FCODATE	Date of File Cabinet reorganization
$FCOTIME	Time of File Cabinet reorganization
$FILE1	Name of input file to be processed by the form
$FILE2	Name of output file when processing by the form is finished
$FILESPEC	VMS file specification of a file being converted from Version 1
$FILESPEC2	VMS file specification of the conversion output file
$FLAG_PAGE	Flag page? Y/N
$FORMNAME	Current form name
$FULLNAME	User's full name

(continued)

Table 8-2 Permanent Symbols (Cont.)

Symbol	Description
$IDOCNUM	Document number shown on MDARG form
$IFOLDER	Folder name on MDARG form after validation
$ITITLE	Title shown on MDARG form
$JOBMSG	Y/N flag
$LAST_PTR_DEST	Destination of print output
$LIBDATE	Batch job start date
$LIBGRADE	Form library name to be converted from Version 1
$LIBRNAME	Name of form library containing current form in Forms Development
$LIBTIME	Batch start time
$MASTER	Current type of master in System Manager menu
$ME_DATE_SAVE	Date of current meeting
$ME_SEL2_SAVE	Not currently used (second selection field)
$ME_TIME_SAVE	Start time of current meeting (first selection field)
$NEWDSAB	New document type when you convert a document from one type to another
$NEXT_LIST_ITEM	Contains the result of the NEXT_LIST function
$NIFULL	Expanded descriptive nickname
$NINAME	Symbolic nickname
OATERMINATOR	Set of form function when called from subprocess
$OA_DCL	Flag indicating whether user is currently in subprocess: 0 — in main process; 1 — in subprocess
$PARSED_USER_NAME	Contains the user's last name

(continued)

Table 8-2 Permanent Symbols (Cont.)

Symbol	Description
$PARSE_USER_NAME	1- to 9-character contraction of the username suitable for file names
$PHNUMBER	Used by CBI; contains demo phone numbers
$PHONE_USER	Used by CBI; contains demo phone numbers
$PH_PER_NUM	Used by CBI; contains demo phone numbers
$PH_SHORT_DIR	Used by CBI; contains the type of phone directory: a (ALL-IN-1), c (corporate) p (personal)
$PRT_AFTER	Specifies a time after which to print
$PRT_FEED	Print using form feed flag for printer setup
$PRT_HDR	Print headers? Y/N
$PRT_PRIO	Print job priority
$P_PHNUMBER	Phone for current personal directory selection
$P_PHONE_USER	Name of current selected personal directory entry
$P_PH_PER_NUM	Phone for current personal directory selection
$RESULT	Result of GET function called from subprocess
$RE_DATE_SAVE	Date of current reminder
$RE_NUMBER_SAVE	Number of current reminder (first selection field)
$SAVE_n	Contents of register n for Desk Calculator; n may be: X (display register), A (register A), B (register B), C (register C), P (decimal-places register)
$SCHCYCLE	Stores frequency of janitor runs

(continued)

Table 8-2 Permanent Symbols (Cont.)

Symbol	Description
$SCHDATE	Stores the date the janitor runs
$SCHTIME	Stores the time the janitor runs
$SELECT	Used for title on Form DOCSEL
$SEPREC_N	Number of years in life of depreciation item
$SMSCH_FOLDER	The name of the folder on which to run the janitor
$SM_ADD	Selects /MODE=ADD for the form
$SM_CHANGE	Selects /MODE=CHANGE for the form
$SM_DELETE	Selects /MODE=DELETE for the form
$SM_INQUIRE	Selects /MODE=INQUIRE for the form
$SM_SEARCH	Form to search for entries
$SM_UPDATE	Selects /MODE=UPDATE for the form
$SRCFORM	Name of form being extracted or the form from which another form is created
$SRCLIBR	Source form library from which to copy
$STDNAME	Current standard form type in Forms Development
$TD_DATE_SAVE	Date last set in to-do list (not used for selection)
$TD_NUMBER_SAVE	Number of current to-do list item
$TOP_FORM	Saves name of previous form before call to File Cabinet from Word and Document Processing
$TRA_A1LOG	Tracing flag: 1=on, 0=off
$TRA_ALL	Tracing flag: 1=on, 0=off
$TRA_DCL	Tracing flag: 1=on, 0=off
$TRA_DPEINI	Tracing flag: 1=on, 0=off

(continued)

Table 8-2 Permanent Symbols (Cont.)

Symbol	Description
$TRA_FIELD	Tracing flag: 1=on, 0=off
$TRA_FLOW	Tracing flag: 1=on, 0=off
$TRA_FORM	Tracing flag: 1=on, 0=off
$TRA_HELP	Tracing flag: 1=on, 0=off
$TRA_INPUT	Tracing flag: 1=on, 0=off
$TRA_IO	Tracing flag: 1=on, 0=off
$TRA_IVP	Tracing flag: 1=on, 0=off
$TRA_LOG	Tracing flag: 1=on, 0=off
$TRA_MESS	Tracing flag: 1=on, 0=off
$TRA_PUT	Tracing flag: 1=on, 0=off
$TRA_SCRCHR	Tracing flag: 1=on, 0=off
$TRA_SCRIPT	Tracing flag: 1=on, 0=off
$TRA_STUMP	Tracing flag: 1=on, 0=off
$TRA_SYMBOL	Tracing flag: 1=on, 0=off
$TRA_TERM	Tracing flag: 1=on, 0=off
$USERNAME	ALL-IN-1 username
$VMS_ACCOUNT	User account information
$VMS_DEVICE	User default device information
$VMS_DIRECTORY	User directory information
$VMS_PASSWORD	User password information
$VMS_QUOTA	User disk quota information
$VMS_UIC	User UIC information
$WPDOC	Current word processing document
$WPFDR	Initial folder name on MDARG form

9

The File Cabinet

The File Cabinet provides tools for storing and retrieving documents. Specifically, this facility provides subfunctions that:

- Store document attributes that give information about each document.
- Organize documents into groups (folders, cabinets).
- Retrieve documents based on attribute values.
- Store a single document in multiple folders, cabinets or attachment lists.
- Create composite documents from lists of documents (attachments).
- Allow documents to be shared by more than one user.
- Allow discarded documents to be recovered.

9.1 File Cabinet Concepts

The following terms describe concepts that are central to the File Cabinet facility.

File Cabinet

The ALL-IN-1 File Cabinet is a data set that consists of:

- Shared system files:
  - OA$DATA:DAF.DAT, the shared Document Attributes file (SDAF)
  - OA$DATA:PENDING.DAT, the system Pending file
- Personal data files for each ALL-IN-1 user:
  - OAUSER:DOCDB.DAT, the personal File Cabinet index file
  - OAUSER:DAF.DAT, the personal Document Attributes file (PDAF)
- Actual RMS files, or documents, that these four data files index

The organization of the File Cabinet is illustrated in Figure 9-1.

Document

A document is an RMS file described by entries in one or more of the File Cabinet index files. The format of a document is, for the most part, independent of how that document is treated by the File Cabinet. That is, most CABINET subfunctions deal with files indirectly, and you can use these subfunctions to manage ASCII text files, indexed data files, binary files or any type of RMS file.

Document Numbers

When you create a document, the File Cabinet gives it a unique identifying number, and combines that number, either as-is or after it is inverted, with the name of the folder in which you store the document. This combination serves as the primary key for document storage and retrieval.

Folders

The File Cabinet stores documents in folders. A folder is simply defined by the documents it contains; documents may be associated with one or more folders.

Documents are usually stored in folders in Last In, First Out (LIFO) order. However, your site may define First In, First Out (FIFO) folders, as covered in the *System Manager's Guide*. Some system-maintained folders are FIFO by default.

Attributes

All documents have attributes. An attribute is a name/value pair that gives information about the document. In the simplest case, a document's attributes correspond to fields and field contents in OAUSER:DOCDB.DAT, a file known only to the user. More complex documents and shared documents have additional attributes, stored in attributes files (the user's DAF.DAT or, for shared documents, OA$DATA:DAF.DAT).

The File Cabinet automatically assigns and updates attributes when you call certain CABINET subfunctions. You can call other subfunctions to selectively assign and update attributes.

Attachments

You can attach any document to any other document to form a new compound document. If you attach additional documents to this compound document, the File Cabinet appends each new attachment to a list of attachments associated with the original document.

If you attach a document that also has attachments, then the File Cabinet treats the attachments as attachments of the original document.

Shared Documents

Entries in a user's personal File Cabinet index may refer to a single physical copy of a document that is shared between a number of users. The File Cabinet stores such shared documents in a common area, and users cannot edit or update them without first making a private copy.

The File Cabinet makes Mail messages into shared documents once they are sent. It also stores document attachments as shared documents.

Shared documents have different document numbers for each user sharing the document.

Janitor

The Janitor empties the Wastebasket folder. After the documents have been emptied from the Wastebasket, they can not be recovered.

These concepts are explained in more detail under the subfunctions that are associated with them.

9.2 File Cabinet Organization

File Cabinet documents range from simple documents known only to one user, to documents that are indexed through several files and shared by many users. The File Cabinet index files are described in the following sections.

9.2.1 OAUSER:DOCDB.DAT, the Personal File Cabinet Index

The personal File Cabinet file is OAUSER:DOCDB.DAT, an RMS indexed file that contains one or more entries, or document headers, for each document that the user has access to. The documents themselves are RMS files stored in directory OAUSER or possibly in a shared area.

DOCDB.DAT is indexed by these keys:

- Key 0 (primary) — The primary DOCDB.DAT key is the first 36 characters of data in the file, corresponding to DOCDB fields FOLDER and FIXER. FOLDER is the 30-character, user- or system-supplied name of the folder in which the document is stored. FIXER is a 6-digit, system-supplied number that is appended to FOLDER to ensure that the document is stored and retrieved in a meaningful and efficient manner.

Key 0 is a unique key: many DOCDB.DAT records may share one FOLDER value (or be stored in one FOLDER), and FIXER may repeat when a document is cross-filed; but each document is assigned a unique FIXER within a given FOLDER. Thus, the FOLDER/FIXER combination stored in field DOCDB.DOCUMENT uniquely identifies a document in the context of DOCDB.DAT. In contrast, the File Cabinet concatenates the system-assigned document number, DOCNUM — which is often the inverse of FIXER — to FOLDER, to uniquely identify a document in CABINET function calls.

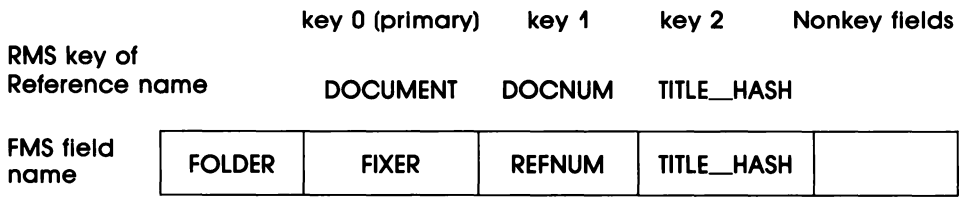
- Key 1 (alternate) — Whenever you create a new document, the File Cabinet assigns it a unique 6-digit document number, DOCNUM, in ascending order. DOCNUM is DOCDB.DAT key 1. DOCNUM allows direct document access by number.

  Key 1 is a nonunique key: you can repeat DOCNUMs, but each FOLDER/DOCNUM combination is unique.

- Key 2 (alternate) — TITLE__ HASH is the first six characters of the user-supplied document title, converted to uppercase to ensure that it is a consistent key of reference. The File Cabinet uses this field to look up documents by title.

  Key 2 is a nonunique key.

The following diagram illustrates this key structure.



The File Cabinet creates and maintains the Wastebasket and Electronic Messages folders. The Messages folders, UNSENT (or CREATED), SENT (or OUTBOX), UNREAD (or INBOX), and READ, are covered in more detail in Chapter 11. Some of these system-maintained FOLDERs influence the value of FIXER.

FIXER always equals either DOCNUM or its inverse, REFNUM. The value assigned depends on the folder. The default File Cabinet action to assign the value of REFNUM to FIXER causes the most recently created document in a folder to display first during searches.

The following table shows how FIXER is determined for specified system-maintained folders and briefly explains the side effects of storing documents in this way.

FOLDER	FIXER equals:	Result
UNSENT	REFNUM; for example, 999998	UNSENT messages presented in LIFO order
SENT	REFNUM; for example, 999998	SENT messages presented in LIFO order
INBOX OR UNREAD	DOCNUM; for example, 000002	UNREAD messages presented in FIFO order
READ	REFNUM; for example, 999998	READ messages presented in LIFO order

The nonkey fields in DOCDB.DAT are:

- FILENAME — This 64-character field contains the RMS file name of the primary text of the document. The Cabinet also uses FILENAME as a pointer to the DAF.DAT files, where it serves as the primary key.
- DAPOINTER — This 1-character flag indicates whether the document is private or shared:
  - When DAPOINTER has a value of P, a record exists for this document in OAUSER:DAF.DAT. The document is known only to the user.
  - When DAPOINTER has a value of S, a record exists for this document in OA$DATA:DAF.DAT.

- TITLE — This 72-character field contains the full title of the document in both upper and lowercase letters. Alternate key TITLE_HASH, described previously, is an abbreviated version of TITLE suitable for key searches.
- AUTHOR — The 30-character name of the author of the document.
- TYPE — This is the usage of the document: DOCUMENT, MAIL, DICTIONARY, and so on. This field is set by the application and is generally not changed by the user.
- SETUP — This 15-character field is currently unused by the File Cabinet and is provided for compatibility with the Version 1.X File Cabinet applications.
- FORMAT — This field contains a keyword that identifies the format, or handling method for the document. This field is also referred to as the HANDLING field. The value in this field corresponds to a key value in OA$DATA:FORMAT.DAT.
- DSAB — This field contains the name of a Text Data Set type. The contents of this field control how the document is handled internally by ALL-IN-1 text processing routines.

  The DSAB is typically one of a limited number of values, such as ASCII, WPSPLUS, or DX. These values correspond to the key values of the TXTFMT format table. This table can be found in the module OAGBL. The values are defined in the column DSAB name.
- KEYWORDS — This 50-character field contains one or more optional, user-supplied keywords that help categorize the document.
- CREDAT — This 8-character creation-date field contains the date that the document was created in mm/dd/yy format.
- MODDAT — This 8-character modification-date field contains the date, in mm/dd/yy format, that the document was last edited or had its header record(s) updated.
- MAIL_ORIG — This field contains the DOCNUM of the original message. It is used to forward answers.
- MAIL_STATUS — This field contains the status of the document if it is a mail message (that is, SENT, UNSENT, and so on).
- DOCNUM — This 6-digit field is the original, system-assigned document number that is generated when the document is created. REFNUM is the inverse of DOCNUM.

Special symbol OA$CURDOC contains the concatenated FOLDER/DOCNUM value for the current document. This symbol, or a value in the same format, is generally used in CABINET function calls to uniquely identify a document.

- DELETE — This 1-character flag indicates whether the document can be deleted from the File Cabinet.
- MODIFY — This 1-character flag indicates whether the user can modify the attributes or the text of the document.
- CREATED — This 17-character field contains the date and time that the document was created in dd-mmm-yyyy hh:mm format.
- MODIFIED — This 17-character field contains the date and time, in dd-mmm-yyyy hh:mm format, that the document or its header record(s) were last modified.

All fields in DOCDB.DAT are considered attributes since they provide some information about a given document. Attributes are covered in the following sections.

9.3 Document Attributes

Treating mail messages and word processing files as two different types of File Cabinet documents allows applications to treat all file-based information with consistency. However, a single, private DOCDB.DAT for each user is not appropriate for storing the information needed when a document is to be shared between users.

The fields of DOCDB.DAT can also be thought of as *attributes* that provide the information about a document that is needed regardless of how the document is to be used. Document-sharing information for a given user is stored in the form of *extended attributes* in one of two files: OAUSER:DAF.DAT and OA$DATA:DAF.DAT. The DAPOINTER field in each OAUSER:DOCDB.DAT record is always nonblank, and it contains a flag that points to one of these two files.

9.3.1 The Document Attributes Files

Each user has a personal document attributes file, OAUSER:DAF.DAT, which contains records for documents that have extended attributes and are not stored in the shared area. For example, when a mail message is created, a number of extended attributes are assigned to the message, and these attributes are recorded in OAUSER:DAF.DAT.

When the SHARE_DOCUMENT subfunction is called, these PDAF attributes are copied to the shared document attributes file (SDAF), OA$DATA:DAF.DAT. New attributes are created based on information in OAUSER:DOCDB.DAT and moved to the SDAF. The text of the message is moved to the shared area. The key of the DAF record changes because the file name in the private area is not unique when moved to the shared area.

The personal and shared DAF.DAT files are functionally extensions of DOCDB.DAT, providing additional information about all shared and compound documents. The PDAF is formatted identically to the SDAF; the main difference is that information in the SDAF is usually read-only by users. Since the SDAF is owned by the system, only certain CABINET subfunctions can modify it.

Both document attributes files are indexed by the RMS file specification of a document, which is stored in field FILENAME of OAUSER:DOCDB.DAT. Thus FILENAME serves as a pointer into the DAF.DAT files.

9.3.2 Document Attributes

Tables 9-1a and 9-1b describe the attributes that a document may have.

Table 9-1a Document Attributes

Attribute	Stored in:	Value
FOLDER	OAUSER:DOCDB.DAT	Folder name; required key-0 segment for DOCDB.DAT
FIXER	OAUSER:DOCDB.DAT	Same as either REFNUM or DOCNUM; required key-0 segment for DOCDB.DAT
REFNUM	OAUSER:DOCDB.DAT	Reference number; key field for DOCDB.DAT
TITLE_HASH	OAUSER:DOCDB.DAT	Uppercased, abbreviated title; optional DOCDB.DAT key
FILENAME	OAUSER:DOCDB.DAT	RMS file name; used as key to the DAF.DAT files
DAPOINTER	OAUSER:DOCDB.DAT	Flag; indicates where attributes are stored
TITLE	OAUSER:DOCDB.DAT	Document title
AUTHOR	OAUSER:DOCDB.DAT	Document author
TYPE	OAUSER:DOCDB.DAT	Document usage
SETUP	OAUSER:DOCDB.DAT	Unused
FORMAT	OAUSER:DOCDB.DAT	Document format (handling)
DSAB	OAUSER:DOCDB.DAT	Text Data Set type for internal handling
KEYWORDS	OAUSER:DOCDB.DAT	Document keywords
CREDAT	OAUSER:DOCDB.DAT	Document creation date, mm/dd/yy
MODDAT	OAUSER:DOCDB.DAT	Document modification date, mm/dd/yy
MAIL_ORIG	OAUSER:DOCDB.DAT	Mail DOCNUM of original message
MAIL_STATUS	OAUSER:DOCDB.DAT	Message status
DOCNUM	OAUSER:DOCDB.DAT	System-assigned document number
DELETE	OAUSER:DOCDB.DAT	Delete flag
MODIFY	OAUSER:DOCDB.DAT	Modify flag
CREATED	OAUSER:DOCDB.DAT	Document creation date, dd-mmm-yyyy hh:mm
MODIFIED	OAUSER:DOCDB.DAT	Document modification date, dd-mmm-yyyy hh:mm

Table 9-1b Extended Attributes

Attribute	Stored in:	Value
USAGE_COUNT	PDAF or SDAF	Number of copies of this document/ message currently outstanding
DR	PDAF or SDAF	Delivery-Receipt-Requested flag
RR	PDAF or SDAF	Read-Receipt-Requested flag
PRIORITY	PDAF or SDAF	Keyword indicating priority for remote mail sending; possible values are: EXPRESS, FIRST_CLASS, and SECOND_CLASS
FORWARDABLE	PDAF or SDAF	Flag indicating whether document/ message is forwardable
TO*	PDAF or SDAF	Name of ALL-IN-1 user to receive the document/message
CC*	PDAF or SDAF	Name of ALL-IN-1 user to receive a carbon copy of the message
SUBJECT	PDAF or SDAF	Subject of the document/message
FROM_ADDRESS	PDAF or SDAF	Sender's address
FROM_DEPT	PDAF or SDAF	Sender's department
FROM_NAME	PDAF or SDAF	Sender's name
FROM_TEL_NO	PDAF or SDAF	Sender's telephone number
DELIVERED	PDAF or SDAF	Date document/message is delivered
POSTED	PDAF or SDAF	Date document/message is posted
FUN*	PDAF or SDAF	Name of ALL-IN-1 function to perform at message receipt
JOB_NUMBER	PDAF or SDAF	Reserved to DIGITAL
ATT_DOC*	PDAF or SDAF	The key to the SDAF for an attached document
MAIL_ID	PDAF OR SDAF	System-generated message ID
BCC*	PDAF or SDAF	Name of ALL-IN-1 user to receive a blind copy of document/message
VOICE_MESSAGE	PDAF or SDAF	Voice-mail message
VOICE_SYSTEM	PDAF or SDAF	Voice-mail system
VOICE_USER	PDAF or SDAF	Voice-mail username
DTITLE	PDAF or SDAF	Document title; same value as DOCDB.TITLE
DTYPE	PDAF or SDAF	Document type; same value as DOCDB.TYPE
DSETUP	PDAF or SDAF	Document setup; same value as DOCDB.SETUP
DFORMAT	PDAF or SDAF	Document format; same as DOCDB.FORMAT

(continued)

Table 9-1b Extended Attributes

Attribute	Stored in:	Value
DCREATED	PDAF or SDAF	Document creation date; same value as DOCDB.CREATED
DMODIFIED	PDAF or SDAF	Document modification date; same value as DOCDB.MODIFIED
DDSAB	PDAF or SDAF	Text data set type; same value as DOCDB.DSAB

* Attributes may occur more than once per document

As shown in Table 9-1, most extended attributes are only meaningful for mail messages. However, any File Cabinet document that is given the correct attributes can be sent, forwarded, and otherwise treated as a message.

9.3.3 The System Pending File

One other file, OA$DATA:PENDING.DAT, contains File Cabinet related information. The pending file is essentially a message board: there is a record for each ALL-IN-1 user, and the record is indexed by the ALL-IN-1 username appended to the application (for example: "MAIL USERNAME"). The messages for that user are stored in the nonkey fields of the record.

The primary user of the pending file is the Mail application. When you send a message, the File Cabinet updates each recipient's pending-file record to reflect that a new unread message is waiting. Since the RMS filename of that pending message is stored in the pending record, the File Cabinet facility knows where to go to get the text of the message when the user decides to read it. The actual message is always stored in the shared area.

Figure 9-1 illustrates the organization of the File Cabinet.

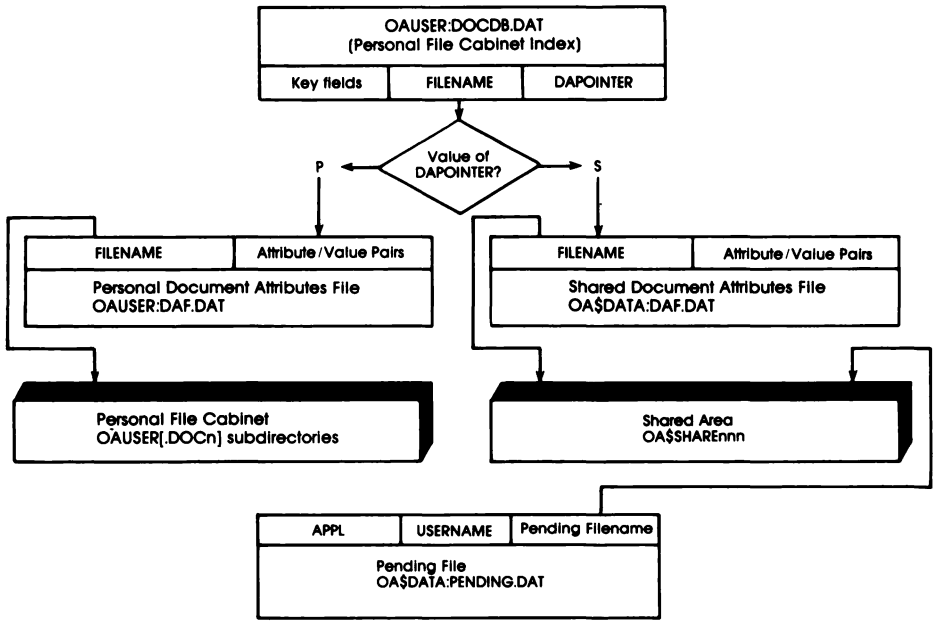


Figure 9-1 File Cabinet Organization

9.4 Using the File Cabinet

9.4.1 Summary of CABINET Subfunctions

The ALL-IN-1 CABINET function takes subfunctions as parameters. These subfunctions are summarized in the following list and covered in detail later in this chapter.

- Creating documents and managing how they are stored:
  - CREATE — Creates a File Cabinet entry for a new document
  - COPY — Makes a physical copy of a file with a new File Cabinet entry
  - REFILE_DOCUMENT — Moves an existing document to a new folder

    OR

    DELETE_OR_REFILE — Moves a document from the orginial folder to the target folder

  - CROSSFILE — Files a document in an additional folder

- Selecting specific documents to work with:
  - CURRENT — Makes a document current
  - SELECT — Finds a document from number, title, folder
  - NEXT_nnn — Finds the next document in a sequence
  - SCAN — Sets up search criteria for documents
- Updating the attributes of a document:
  - ADD_ATTRIBUTE — Adds a new attribute value
  - CHANGE_ATTRIBUTE — Changes the value of an attribute
  - DELETE_ATTRIBUTE - Deletes the value of an attribute
  - EDIT_DOCUMENT — Changes the modified date of the document
  - RESET_DELETE — Changes the value of the Delete Flag (OAUSER:DOCDB.DAT)
- Controlling the buffering of attribute updates:
  - BEGIN — Starts updating a document without actually modifying the Attributes file
  - END — Completes the period of deferred update started by BEGIN and updates the Attributes file
  - FORCE — Forces immediate updates while in a BEGIN/END context
- Arranging for local delivery of mail messages:
  - GET_PENDING — Gets an entry from the pending file
  - PUT_PENDING — Adds an entry to the pending file
  - SHARE_DOCUMENT — Makes a document a shared document
  - UNSHARE_DOCUMENT — Makes a shared document unshared
- Manipulating document attachments to mail messages:
  - ATTACH_DOCUMENT — Attaches a document
  - DETACH_DOCUMENT — Detaches a document
  - FILE_ATTACHMENT — Files an attachment document

- Deleting documents:
  - DELETE_DOCUMENT — Deletes a document from File Cabinet
  - JANITOR — Empties the contents of the WASTEBASKET folder
- Miscellaneous
  - CLEAR_CURDOC — Clears the value of symbol OA$CURDOC
  - CLOSE — Closes the File Cabinet
  - CONVERT — Converts Version 1.X File Cabinet to Version 2.0
  - DECREMENT_COUNT — Modifies the count field of the pending record
  - EXCHANGE — Exchanges the value of OA$CURDOC with the document on the top of the document stack
  - HELP — Provides a list of File Cabinet functions
  - OPEN — Opens the File Cabinet files
  - POP — Pops the top document from the stack to OA$CURDOC
  - PUSH — Pushes the current document onto the Stack
  - REORG_DOCDB — Reorganizes a user's personal File Cabinet files.
  - STOP_SCAN — Stops a CABINET SCAN

The File Cabinet records the status of every CABINET subfunction call in special symbol OA$STATUS. A TRUE value (low-order bit is 1) indicates that the subfunction call was completed sucessfully, and a FALSE value (low order bit is 0) indicates that the call was unsuccessful.

9.4.2 Accessing File Cabinet Data

Simple data sets usually consist of an entry form and a related data file. The File Cabinet is a complex data set that encompasses several data files and is accessed through a number of CABINET subfunctions. Because these files are so closely interrelated, they are not always accessable in the standard ways (entry forms, WRITE function calls, and so forth).

You maintain all File Cabinet related files with calls to CABINET subfunctions. You can retrieve or compare against information from all File Cabinet files using a special data set reference (DSR) syntax.

Since ALL-IN-1 treats DSRs as read-only symbols, you can retrieve File Cabinet based information wherever ALL-IN-1 symbols are recognized: as function parameters, field-qualifier arguments, and so on. You cannot use File Cabinet DSRs to modify File Cabinet files: for example, specifying:

```
/PUT = CAB$.TITLE[$WPDOC]
```

OR

```
<GET CAB$.FORMAT[OA$CURDOC] = "RUNOFF"
```

results in an error. As mentioned previously, CABINET subfunctions perform these tasks.

File Cabinet DSRs that reference the entire File Cabinet have the following general syntax:

```
CAB$.attribute_name[key_sym]
```

CAB$ identifies the entire File Cabinet data set. Use this syntax to refer to the value of a single attribute, regardless of where that attribute is stored (OAUSER:DOCDB.DAT, OAUSER:DAF.DAT, or OA$DATA:DAF.DAT). Only one value for a repeating attribute (such as the TO attribute) can be accessed via the CAB$ syntax.

The key_sym in square brackets is a FOLDER/DOCNUM combination that has the same format as special symbol OA$CURDOC.

For example, to get the RMS file name of the current document:

```
<GET CAB$.FILENAME[OA$CURDOC]
```

To examine the TITLE fields of all documents:

```
<FOR CAB$ DO PROMPT .TITLE
```

CAB$ATTRIBUTES:attribute_sym.VALUE[key_sym]
CAB$ATTRIBUTES:attribute_sym.KEY

CAB$ATTRIBUTES refers to the value(s) of attribute(s) stored in one of the attributes files, OAUSER:DAF.DAT or OA$DATA:DAF.DAT. CAB$ATTRIBUTES provides a mechanism for scanning repeating attributes.

The attribute to retrieve is identified by attribute_sym, and the syntax of the rest of the DSR depends on what you want to reference. Specify VALUE with a bracketed key_sym to return the value of attribute_sym for the referenced document.

For example:

Get the value of an attribute, or the first value of an attribute with multiple values:

```
<GET CAB$ATTRIBUTES.VALUE["TO"]
```

To interactively process all TOs for all documents:

```
<FOR CAB$ATTRIBUTES:"TO" DO ...
```

To process all attributes for the current document as a menu option from Named Data:

```
1 Name ALL
FOR CAB$ATTRIBUTES DO ...
```

To interactively view all attributes:

```
<FOR CAB$ATTRIBUTES DO PROMPT .KEY ": " .VALUE
```

CAB$ATTACH.attribute_sym[key_sym]

CAB$ATTACH accesses the attachments of the current document. Specifically, it returns the keys to the SDAF of the attached documents, allowing you to reference the attributes of each attachment through another DSR, CAB$ATTACH_ATTRIBUTES.

For example:

```
<FOR CAB$ATTACH DO GET .DTITLE
```

This opens the CAB$ATTACH data set and returns the titles of all attachments.

```
<FOR CAB$ATTACH DO GET .VALUE
```

This returns the keys to the SDAF for all attachments.

NOTE: Where there are multiple values to be displayed, a full listing can be obtained by pressing Gold W.

CAB$ATTACH_ATTRIBUTES

This DSR accesses the attributes of the attachments specified by CAB$ATTACH. You can use CAB$ATTACH_ATTRIBUTES with the following nested FOR function call:

```
<FOR CAB$ATTACH DO FOR CAB$ATTACH_ATTRIBUTES DO GET .KEY ": ".VALUE
```

This single nested function call:

1. GETs the default key of the first attachment by calling CAB$ATTACH for attribute ATT_DOC.
2. GETs the attributes of the first attachment by calling CAB$ATTACH_ATTRIBUTES.
3. Repeats these two steps to position the DSRs to the next attached document, as long as there are attachments.

Frequently used File Cabinet information is also stored in special ALL-IN-1 symbols. Special symbols return information faster than DSRs and should be used in preference to DSRs. The OA$CURDOC_XXX special symbols, including OA$STATUS, are automatically maintained by the CABINET subfunctions. Table 8-1 lists these special symbols, their descriptions and their equivalents. The OA$CURMES_XXX special symbols, which are maintained by MAIL subfunctions, also store Cabinet related data and are likewise listed in Table 8-1.

9.4.3 The Shared Area

When you or the File Cabinet designate a document as shared, the File Cabinet moves it to a common directory. This common directory is most often one of a series of parallel directories. A series of system logical names identify these directories and determine where the File Cabinet will store the next shared document.

The current shared directory(that is, the area where the next shared document is stored) is indirectly identified by the system logical OA$SHARE_CUR. OA$SHARE_CUR contains a 1- to 3-digit number, assigned by the File Cabinet, that identifies one of up to 1,000 other logicals. This other logical is the one that the Cabinet actually associates with the current shared directory.

The ALL-IN-1 system manager usually creates and maintains the logicals that OA$SHARE_CUR points to. These logicals have the general format OA$SHAREnnn, where the value of OA$SHARE_CUR corresponds to nnn. The File Cabinet maintains OA$SHARE_CUR to ensure that all shared directories contain a roughly equal number of documents.

The *System Manager's Guide* contains a more detailed discussion of the shared area.

9.4.4 Shared Libraries

A shared library is a collection of one or more folders in a user's ALL-IN-1 File Cabinet. The user makes these folders available for other ALL-IN-1 users.

The shared library concept has several important restrictions:

- Shared library documents may only be accessed for READ operations. Only the owner may modify the shared library document.
- The VMS file protection codes establish access privileges and protection. Thus, document files and file-cabinet directories must have the appropriate READ protection in order for the actual read operation to take place. Since there is no ALL-IN-1 function available in V2.0 to modify the VMS file protection of a document,

the System Manager must establish the proper file protection for shared libraries through the default file protection of the owner's VMS account.

- The shared library and all of its users must reside on the same network node or cluster.

The number of shared libraries is up to each individual ALL-IN-1 site. Usually, the system manager establishes a single shared library with multiple folders to classify the shared documents. Because of the protection mechanism which controls access to shared documents, an account which establishes a shared library should only be used for that purpose. See the *System Manager's Guide* for more details.

In group shared libraries the System Manager can combine users within the same VMS UIC group and set the default group file protection for the shared library account accordingly. This requires careful management by the VMS System Manager and is only as flexible as the VMS V3 grouping capability currently available. For this reason, if group shared libraries are not essential, it is *not* recommended that they be established.

A user may access a shared library at any time, to read a document or make a copy of it, but may not move the document. You access the shared library folder using the same folder name that the owner of the shared library document used when he created the folder. The syntax is:

[username]folder-name

For example, if a user named Jack Jones creates a shared library document called HELP, another user can access that shared folder and any document in that folder by typing in the FOLDER field of the Selection form:

[JONES]HELP

9.5 CABINET Subfunctions

Following are the CABINET subfunctions. A special symbol, OA$STATUS, contains the status of each call to a subfunction. For example, if a CABINET subfunction call does not find a specified document to be selected, it sets OA$STATUS to FALSE (low-order bit of 0) to indicate failure. All successful calls set OA$STATUS to TRUE.

ADD_ATTRIBUTE

This subfunction adds an attribute to the specified document. The attribute may be an OAUSER:DOCDB.DAT field or a field in the Document Attributes files. You can only call ADD_ATTRIBUTE if the value of field DOCDB.MODIFY is "Y".

The format is:

```
CABINET ADD_ATTRIBUTE [doc-sym], attr-sym, val-sym
```

where

doc-sym is an ALL-IN-1 symbol representing the FOLDER/DOCNUM combination of the document to which to add an attribute. This parameter is optional; when you omit it, ADD_ATTRIBUTE uses the value of OA$CURDOC.

attr-sym is an ALL-IN-1 symbol representing the name of the attribute to add. This parameter may be a DOCDB.DAT field name (such as TITLE, MODDAT) or the name of an extended attribute (such as TO).

val-sym is an ALL-IN-1 symbol representing the value of the attribute.

For example:

```
<CAB ADD_ATTRIBUTE $WPDOC, "TO","JENSEN"
```

This interactive function call adds a TO attribute, with the value JENSEN, to the document represented by $WPDOC. This attribute value is stored in file OAUSER:DAF.DAT.

```
<CAB ADD_ATTRIBUTE $WPDOC, "AUTHOR",OA$USER
```

This interactive call makes the current ALL-IN-1 user the author of the current document.

ATTACH_DOCUMENT

The ATTACH_DOCUMENT subfunction attaches a document to another document. The syntax is:

```
CABINET ATTACH_DOCUMENT [target-doc-sym], attach-sym
```

where

target-doc-sym — is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to which to attach. When omitted, this parameter defaults to the value of OA$CURDOC.

attach-sym — is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to attach.

CABINET ATTACH_DOCUMENT does one of two things, depending on whether the attachment is already shared. If the attachment is already shared, the SDAF record for the attachment is updated. If the attachment is not already shared, CABINET ATTACH_DOCUMENT calls the SHARE subfunction internally for the attachment.

In both cases, ATTACH_DOCUMENT increments the value of the attachment's USAGE_COUNT attribute in the SDAF.

You can refile a document attached to another document as a separate document. This refiling has no effect on the original composite document, which still retains all its attachments. You can still edit the attached document unless the MODIFY flag is set to N.

You can also detach attached documents with the DETACH_DOCUMENT subfunction.

BEGIN

CABINET BEGIN improves performance when called with other subfunctions. Calling this subfunction causes the File Cabinet facility to defer updating to the PDAF until:

— You call CABINET END.

OR

— You call CABINET FORCE.

CABINET BEGIN takes no parameters. Several other subfunctions call CABINET FORCE implicitly.

CHANGE_ATTRIBUTE

This subfunction changes the value of an existing attribute of the specified document. You cannot use CABINET CHANGE_ATTRIBUTE to change the filename of a document. You can only call CABINET CHANGE_ATTRIBUTE if the DOCDB.MODIFY field is set to Y.

The syntax is:

CABINET CHANGE_ATTRIBUTE [doc-sym], attr-sym, old-val-sym, new-val-sym

where

doc-sym — is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document containing the attribute. This parameter is optional and defaults to the value of OA$CURDOC if omitted.

attr-sym — is an ALL-IN-1 symbol representing the name of the attribute to alter. This parameter may be a DOCDB.DAT field name (such as TITLE, MODDAT) or the name of an extended attribute (such as TO).

old-val-sym — is an ALL-IN-1 symbol representing the old value of the attribute.

new-val-sym — is an ALL-IN-1 symbol representing the new value of the attribute.

CLEAR_CURDOC

Call this function to clear the contents of the ALL-in-1 symbol OA$CURDOC. The File Cabinet calls this function automatically whenever you retrieve the value of OA$CURDOC with the GET function, the /GET form qualifier, or the /GET_SAVE field qualifier.

CLOSE

Use CABINET CLOSE to explicitly close the File Cabinet files. The next call to a CABINET subfunction (other than CLOSE) automatically reopens the File Cabinet.

You can use CABINET CLOSE while developing a File Cabinet related application to temporarily close files and (for example) dump their contents.

CONVERT

The CONVERT subfunction converts OAUSER:DOCDB.DAT records in ALL-IN-1 Version 1.X to the format used by the Version 2.0 File Cabinet. This a one-time-only conversion.

COPY

The COPY subfunction creates a new DOCDB.DAT entry for a document and copies the RMS file and all attributes. The newly created document becomes the current document: that is, its FOLDER/DOCNUM is assigned to OA$CURDOC.

The syntax is:

```
CABINET COPY [doc-sym], newfldr-sym, [trgt-sym], [titl-sym] [,newfil-sym]
```

where

doc-sym — is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to be copied. When omitted, this parameter defaults to the value of OA$CURDOC.

newfldr-sym — is an ALL-IN-1 symbol representing the folder name in which to create the new document.

trgt-sym — an optional parameter, is an ALL-IN-1 symbol which receives the FOLDER/DOCNUM of the newly created document.

titl-sym — an optional parameter, is an ALL-IN-1 symbol representing the TITLE to be given to the newly created document.

newfil-sym — an optional parameter, is an ALL-IN-1 symbol representing the RMS file name of the newly created document. When omitted, ALL-IN-1 creates a file name based on the current date and time.

You can copy both private and shared documents. However, the newly created document is always private.

CABINET COPY does the following:

1. Generates a new, unique DOCNUM.
2. Stores the DOCNUM, REFNUM, FIXER, and FOLDER in the current DOCDB record.
3. Calls the CREATE function to create a new document using the folder name specified in newfldr-sym.
4. Copies the attributes (if any),from the original document's PDAF record (or SDAF record, if the document is shared) to the new document's PDAF record. The value of newfil-sym is used as the key field in the PDAF for the new copy.

5 Treats attachments as attachments of the new copy, and their SDAF USAGE_COUNTs are incremented.

6 Gives the trgt-sym parameter (if supplied), the FOLDER/DOCNUM of the newly created document.

7 OA$CURDOC is assigned the FOLDER/DOCNUM of the newly created document, making it the current document.

CREATE

CABINET CREATE creates an entry for a new document in the File Cabinet; it does not create a file. The syntax is:

```
CABINET CREATE [folder-sym], [filename-sym] [,target-sym]
```

where

folder-sym — is an ALL-IN-1 symbol representing the folder in which the document is created. When omitted, this parameter defaults to the folder associated with permanent symbol $DEFAULT_FOLDER.

filename-sym — an optional parameter, is an ALL-IN-1 symbol that contains the RMS file name to use for the new document. If omitted, the File Cabinet creates a unique file name based on the current date and time.

target-sym — an optional parameter, is an ALL-IN-1 symbol to receive the FOLDER/DOCNUM of the newly created document.

CABINET CREATE does the following:

1 Ensures that the RMS file name is unique. If it is, the Cabinet generates a new DOCNUM and creates a new DOCDB.DAT record.

2 Assigns values to the following DOCDB.DAT fields:

FOLDER
FIXER
REFNUM
DOCNUM
FILENAME
DAPOINTER (value of P, indicating PDAF)
CREDAT/CREATED (current date and time)
MODDAT/MODIFIED (current date and time)
MODIFY (modifiable)
DELETE (deletable)

3 Makes the newly created document the current document by updating OA$CURDOC.

CROSSFILE

The CABINET CROSSFILE subfunction creates another reference to a document by adding a new DOCDB.DAT entry with a different FOLDER name. CROSSFILE creates another DOCDB record that points to the same document: all fields except FOLDER (and optionally TITLE) are copied intact to the new record.

The syntax is:

```
CABINET CROSSFILE doc-sym, folder-sym, [target-sym] [,title-sym]
```

where

doc-sym — is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to cross-file. When omitted, this parameter defaults to the value of OA$CURDOC.

folder-sym — is an ALL-IN-1 symbol representing the folder in which the document is cross-filed.

target-sym — an optional parameter, is an ALL-IN-1 symbol to receive the FOLDER/DOCNUM of the cross-file reference. When you specify target-sym, the new cross-file reference to the document becomes the current document: that is, the new FOLDER/DOCNUM is assigned to OA$CURDOC as well as to target-sym. In all cases, OA$CURDOC contains the FOLDER/DOCNUM of the resultant document.

title-sym an optional parameter, is an ALL-IN-1 symbol representing the TITLE to give to the cross-file reference.

A cross-filed document is still treated as a single document. If you edit the document or update its attributes, the changes occur in all cross-filed entries.

NOTE: *CABINET CROSSFILE sets OA$STATUS to two when you try to cross-file a document in a folder where it already exists.*

CURRENT

The CURRENT subfunction makes the specified document current by updating special symbol OA$CURDOC with the specified parameter. The syntax is:

```
CABINET CURRENT [doc-sym]
```

where doc-sym is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to be made current.

If the named document is already current, CABINET CURRENT does nothing.

You can also call CABINET SELECT to make a document current.

DECREMENT_COUNT

The DECREMENT_COUNT subfunction decrements the value of the extended USAGE_COUNT attribute of the specified user's application.

The syntax is:

```
CABINET DECREMENT_COUNT application_sym
```

where application_sym is an ALL-IN-1 symbol representing the ALL-IN-1 application (for example, Mail) for which the USAGE_COUNT is decremented.

DELETE_ATTRIBUTE

DELETE_ATTRIBUTE deletes an attribute from the specified document. You can only call this subfunction successfully if the MODIFY flag is Y. The attributes can be stored in either OAUSER:DOCDB.DAT or in the Document Attributes file.

The syntax is:

```
CABINET DELETE_ATTRIBUTE [doc-sym], attr-sym, val-sym
```

where

doc-sym is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document containing the attribute to be deleted. This parameter is optional and defaults to OA$CURDOC if omitted.

attr-sym is an ALL-IN-1 symbol representing the attribute to delete. This parameter may be a DOCDB.DAT field name (such as TITLE, MODDAT) or the name of an extended attribute (such as TO).

val-sym is an ALL-IN-1 symbol representing the value of the attribute to delete. In cases where an attribute may occur more than once, a wildcard character, "*", may be specified as the val-sym to delete all occurences of the attribute.

DELETE_DOCUMENT

CABINET DELETE_DOCUMENT deletes a document reference from the File Cabinet. If there are no other references to the document, then DELETE_DOCUMENT also deletes the RMS file associated with the document. The JANITOR subfunction generally calls DELETE_DOCUMENT to physically remove a document.

The syntax is:

```
CABINET DELETE_DOCUMENT [doc-sym]
```

where doc-sym is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to be deleted. When you omit this parameter, DELETE_DOCUMENT deletes the current document.

DELETE_OR_REFILE

CABINET DELETE_OR_REFILE moves a document from the original folder to the target folder. If a copy of the document already exists in the target folder, DELETE_OR_REFILE makes no changes other than to remove it from the source folder.

The syntax is:

```
CABINET DELETE_OR_REFILE [doc-sym], target-folder-sym, [target-sym] [,title-sym]
```

where

doc-sym	is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to delete or refile. DELETE_OR_REFILE always removes this document from the original folder.
target-folder-sym	is an ALL-IN-1 symbol representing the folder in which the document is refiled.
trgt-sym	an optional parameter, is an ALL-IN-1 symbol to receive the FOLDER/DOCNUM of the refiled document. When target-sym is specified, the new refiled reference to the document becomes the current document: that is, the new FOLDER/DOCNUM is assigned to OA$CURDOC as well as to target-sym. In all cases, OA$CURDOC contains the value of the resultant document.
title-sym	an optional parameter, is an ALL-IN-1 symbol that represents the title of the resultant document.

DETACH_DOCUMENT

The DETACH_DOCUMENT subfunction removes an attachment from a document. The syntax is:

```
CABINET DETACH_DOCUMENT [doc-sym], attachment-sym
```

where

doc-sym	is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document from which the attachment is to be removed. When omitted, this parameter defaults to the value of OA$CURDOC, the current document.
attachment-sym	is an ALL-IN-1 symbol representing the key (filename) of the attachment.

EDIT_DOCUMENT

CABINET EDIT_DOCUMENT updates the document's DOCDB.DAT MODDAT and MODIFIED fields to reflect the current system date and time. This date need not be the RMS file's modified date.

The syntax is:

CABINET EDIT_DOCUMENT [doc-sym]

where doc_sym is the FOLDER/DOCNUM combination of the document record to update. When omitted, this parameter defaults to the value of OA$CURDOC.

END

Call CABINET END in conjunction with BEGIN to improve performance during other CABINET subfunction calls. When you call END, the Cabinet performs updates that it had deferred when you called CABINET BEGIN.

CABINET END takes no parameters.

EXCHANGE

CABINET EXCHANGE exchanges OA$CURDOC with the document on the top of an internally maintained document stack. See also the POP and PUSH subfunctions.

CABINET EXCHANGE takes no parameters.

FILE_ATTACHMENT

CABINET FILE_ATTACHMENT creates a new File Cabinet document based on the information supplied. The syntax is:

CABINET FILE_ATTACHMENT [doc-sym], attach-sym, folder-sym [,title-sym]

where

doc-sym — is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to which the attachment is attached. This parameter is optional and defaults to the value of OA$CURDOC when omitted.

attach-sym is an ALL-IN-1 representing the DAF key of the attached file. This is also the value of an ATT_DOC attribute for the parent document.

folder-sym is an ALL-IN-1 symbol representing the folder in which to file the attachment.

title-sym is an ALL-IN-1 symbol representing the new title of the attached file.

FORCE

Use the FORCE subfunction to force immediate updates of DOCDB.DAT and/or the PDAF from within a CABINET BEGIN / CABINET END sequence. Several CABINET functions call CABINET FORCE internally, and you can call this function explicitly when you need to immediately reference a new attribute while in a BEGIN/END sequence.

FORCE takes no parameters.

GET_PENDING

The GET_PENDING function reads from the system pending file and creates DOCDB records in the caller's File Cabinet. The syntax is:

```
CABINET GET_PENDING appl-sym, folder-sym
```

where

appl-sym is an ALL-IN-1 symbol representing the application code for which to retrieve pending entries; for example, MAIL.

folder-sym is an ALL-IN-1 symbol representing the folder to receive pending records; for example, UNREAD.

GET_PENDING does the following:

- Appends the value of appl-sym to the ALL-IN-1 username to form a lookup key for the pending file
- For each entry in PENDING.DAT, GET_PENDING:
  - Reads the SDAF, using the contents of the pending entry as the key

- Simulates the CREATE subfunction to create DOCDB records for each pending entry in the folder specified in the GET_PENDING call
- Gets the following attributes from the SDAF for each pending entry and stores them in the corresponding fields in the OAUSER:DOCDB.DAT
- *DOCDB.DAT Field* *SDAF Attribute*

TYPE	DTYPE
SETUP	DSETUP
TITLE	DTITLE
AUTHOR	DAUTHOR
FORMAT	DFORMAT
DSAB	DDSAB
CREATED	DCREATED
MODIFIED	DMODIFIED

- Gives the DAPOINTER field for each newly created DOCDB.DAT record the value "S" (for SHARE)

- When the appl_sym parameter is MAIL, GET_PENDING:
  - Updates special symbol OA$MAIL_COUNT to reflect the number of new DOCDB.DAT records created by the call
  - Sets the DOCDB.DAT MAIL_STATUS attribute to UNREAD

  Getting the value of special symbol OA$MAIL_COUNT triggers an automatic call to GET_PENDING. The value of this symbol is usually loaded into a field on a menu through the /MAIL form qualifier.

MAIL subfunctions are generally the only callers to GET_PENDING. However, you can call this subfunction to get pending file information for any application that uses this file.

HELP

CABINET HELP lists the CABINET subfunctions.

JANITOR

The CABINET JANITOR subfunction deletes documents from the user's Wastebasket folder or the specified folder. The syntax is:

```
CABINET JANITOR [folder-sym]
```

where folder-sym is an ALL-IN-1 symbol representing the folder to be treated as the Wastebasket folder. When omitted, this parameter defaults to the value of special symbol OA$WASTEBASKET.

CABINET JANITOR does the following:

1. Sequentially reads each document in the named folder, ignoring any document with an "N" in the OAUSER:DOCDB.DAT DELETE field and deleting all other DOCDB records.
2. Using key 1 (REFNUM), attempts to find any remaining DOCDB records. If any are found, the JANITOR reduces the number of references to the document's .DAT record, deletes this DOCDB.DAT record, and is finished with this document.
3. Calls the DETACH DOCUMENT function to remove all attached documents.
4. Checks the value of DAPOINTER for the deleted DOCDB record. If it equals P, JANITOR deletes all records with the key of FILENAME in the PDAF and deletes all versions of FILENAME (the RMS file).
5. Decrements the SDAF usage count field by 1 if DAPOINTER equals S (meaning that the document is shared). If the resultant usage count is 0, then the SDAF record is deleted, along with all versions of FILENAME (the RMS file).

NEXT_DOCUMENT

CABINET NEXT_DOCUMENT returns the next document in the same folder as the specified document. NEXT_DOCUMENT updates the value of OA$CURDOC and thus establishes a new current document. The syntax is:

```
CABINET NEXT_DOCUMENT [doc-sym] [,target-sym]
```

The value of doc-sym may correspond to either a 30-character FOLDER name or a full 36-character FOLDER/DOCNUM value. The optional target-sym parameter, when present, is an ALL-IN-1 symbol that receives the value of the next document. (The value of target-sym remains unchanged if NEXT_DOCUMENT does not find a next document.)

Also, when the function finds the next document, both doc-sym and target-sym contain the same value; and, you can use either one as the doc-sym in a subsequent call to NEXT_DOCUMENT. That is, once the File Cabinet gives these parameters a value, you can use that value to maintain the stream established for NEXT_DOCUMENT.

The following graph illustrates valid doc-sym combinations:

doc-sym		CAB NEXT_DOC returns:
FOLDER	DOCNUM	
—	—	First doc. in cabinet
—	000001	invalid
UNREAD	—	First doc. in UNREAD
UNREAD	000001	Next doc. after "1" in UNREAD

Assume the File Cabinet contains two folders and four documents:

Folder	*Number*
GENERAL	3
GENERAL	2
GENERAL	1
PERSONAL	1

The following example illustrates the results of calling CABINET NEXT_DOCUMENT with different doc-sym combinations:

doc-sym value				target-sym value		
Before call:		*After call:*		*After call:*		*Next document:*
GENERAL	*3*	*GENERAL*	–	*GENERAL*	*3**	"Reached end of folder GENERAL"
GENERAL	2	GENERAL	3	GENERAL	3	Third document in folder GENERAL
GENERAL	1	GENERAL	2	GENERAL	2	Second document in folder GENERAL
GENERAL	–	GENERAL	1	GENERAL	1	First document in folder GENERAL
PROGRAMS	–	PROGRAMS	–	GENERAL	3*	"Folder PROGRAMS is empty"
PERSONAL	5	PERSONAL	–	PERSONAL	1*	"Reached end ßßß"
PERSONAL	1	PERSONAL	–	PERSONAL	1*	"Reached end of folder PERSONAL"
PERSONAL	–	PERSONAL	1	PERSONAL	1	"First document in folder PERSONAL"
MISC	4	MISC	–	PERSONAL	1*	"Folder MISC is empty"
MISC	–	MISC	–	PERSONAL	1*	"Folder MISC is empty"
–	–	GENERAL	1	GENERAL	1	First document in cabinet

*Target-sym remains unchanged; informational message is issued.

NEXT FOLDER

CABINET NEXT_FOLDER finds the next folder after a specified one and makes the newest document in that next folder the current document. The syntax is:

```
CABINET NEXT_FOLDER [doc-sym] [,target-sym]
```

where doc-sym is an ALL-IN-1 symbol that represents the folder to consider current, and target-sym is an ALL-IN-1 symbol that receives the FOLDER/DOCNUM value of the selected document.

When you specify a folder, NEXT_FOLDER looks for the next folder alphabetically. When you omit this parameter, NEXT_FOLDER selects the newest document in the first folder in the File Cabinet. When you include target-sym, NEXT_FOLDER loads the value of OA$CURDOC into that symbol.

If the first parameter is the last folder in the File Cabinet, then target-sym is not modified, and OA$STATUS is set to 0.

The following graph illustrates valid doc-sym combinations:

doc-sym		CABINET NEXT_FOLDER returns:
FOLDER	DOCNUM	
—	—	First doc. in Cabinet
READ	XXXXXX	First doc. after READ

Assume the File Cabinet contains two folders and four documents:

Folder	Number
READ	7
READ	2
READ	1
STATUS	5

The following example illustrates the results of calling CABINET NEXT_FOLDER with different doc-sym combinations:

doc-sym value				target-sym value		
Before call:		After call:		After call:		Next document:
—	—	READ	7	READ	7	First document in cabinet
READ	7	STATUS	5	STATUS	5	First document after READ
STATUS	5	STATUS	5	STATUS	5*	"End of File Cabinet"

*Target-sym remains unchanged; informational message is issued.

NEXT_NUMBER

CABINET NEXT_NUMBER finds the next folder that contains the specified (or current) cross-filed document. NEXT_NUMBER updates the value of OA$CURDOC and thus establishes a new current document. The syntax is:

CABINET NEXT_NUMBER [doc-sym] [,target-sym]

The value of doc-sym may correspond to either a 6-digit DOCNUM or a full 36-character FOLDER/DOCNUM value. The optional target-sym parameter, when present, is an ALL-IN-1 symbol that receives the value of the next document. (The value of target-sym remains unchanged if NEXT_NUMBER does not find a next document.)

Also, when NEXT_NUMBER finds a next document both doc-sym and target-sym contain the same value, and you can use either one as the doc-sym in a subsequent call to NEXT_NUMBER. That is, once the Cabinet gives these parameters a value, you can use that value to maintain the stream established for NEXT_NUMBER.

The following graph illustrates valid doc-sym combinations:

doc-sym		CAB NEXT_NUMBER returns:
FOLDER	DOCNUM	
—	—	Newest doc. in cabinet
—	000001	First folder with this #
MISC	—	invalid
MISC	000001	Next folder with this #

Assume the File Cabinet contains three folders and three documents:

Folder	*Number*
GENERAL	2
GENERAL	1*
PERSONAL	4
PERSONAL	1*
MISC	1*

*When documents are cross-filed or refiled, they retain their original DOCNUM. Thus, GENERAL 1, PERSONAL 1, and MISC 1 refer to the same document in different folders.

Calling CABINET NEXT_NUMBER with different doc-sym combinations results in:

doc-sym value				target-sym value		
Before call:		*After call:*		*After call:*		*Next document:*
—	—	PERSONAL	4	PERSONAL	4	Newest document in cabinet
—	1	GENERAL	1	GENERAL	1	First folder contains doc 1
GENERAL	1	PERSONAL	1	PERSONAL	1	Next folder contains doc 1
PERSONAL	1	MISC	1	MISC	1	Next folder contains doc 1
MISC	1	—	1	MISC	1*	"No other cross-filings"
—	5	—	5	MISC	1*	"Document 5 not found"
—	2	GENERAL	2	GENERAL	2	First folder contains doc 2

*Target-sym remains unchanged; informational message is issued.

NEXT_SCAN

CABINET NEXT_SCAN uses the scanning parameters specified in the most recent CABINET SCAN call to determine the next document. The syntax is:

```
CABINET NEXT_SCAN [doc-sym] [,target-sym]
```

The value of doc-sym must correspond to a full 36-character FOLDER/DOCNUM value. The optional target-sym parameter, when present, is an ALL-IN-1 symbol that receives the value of the next document. (The value of target-sym remains unchanged if NEXT_SCAN does not find a next document.)

Also, when NEXT_SCAN finds a next document, both doc-sym and target-sym contain the same value, and you can use either one as the doc-sym in a subsequent call to NEXT_SCAN. That is, once the Cabinet gives these parameters a value, you can use that value to maintain the stream established for NEXT_SCAN.

Assume the File Cabinet consists of the following five folders and five documents:

Folder	*Number*	*Title*	*Setup*
GENERAL	2	Red	MEMO
GENERAL	1	Blue	MEMO
PROGRAMS	7	Green	SPEC
PERSONAL	2	Orange	MEMO
MISC	5	Green	MEMO
PROJECTS	3	Gray	SPEC

CABINET SCAN is called as follows, prior to calling the first CABINET NEXT_SCAN:

```
CABINET SCAN .SETUP = "MEMO" AND NOT .TITLE = "Orange"
```

Note that SCAN does not select any documents — it simply specifies scanning criteria, which remain in effect until another CABINET SCAN call updates them.

The following example illustrates the results of calling CABINET NEXT_SCAN with different doc-sym values:

doc-sym value				target-sym value		
Before call:		After call:		After call:		NEXT SCAN action:
—	—	GENERAL	1	GENERAL	1	First document matches; selected as next document
GENERAL	1	GENERAL	2	GENERAL	2	Next document matches; selected as next document
GENERAL	2	MISC	5	MISC	5	"PROGRAMS 7" and "PERSONAL 2" skipped; "MISC 5" selected as next document
MISC	5	—	—	MISC	5*	"PROJECTS 3" skipped; "End of scan"

* Target-sym remains unchanged; informational message is issued; doc-sym reset to wraparound to beginning of cabinet.

NOTE: CABINET SCAN and NEXT_SCAN are independent subfunctions. Calling SCAN creates new scanning criteria, but when you call NEXT_SCAN, NEXT_SCAN always resumes scanning where it left off, regardless of whether SCAN has reset the criteria in the meantime. Therefore, if you want to begin NEXT_SCAN searches at the beginning of the File Cabinet, you should always clear the symbol used for the NEXT_SCAN doc-sym parameter after calling SCAN.

NEXT_TITLE

CABINET NEXT_TITLE finds the next document containing the specified title. NEXT_TITLE updates the value of OA$CURDOC and thus establishes a new current document. The syntax is:

```
CABINET NEXT_TITLE [doc-sym], [target-sym], title-sym
```

where the required title-sym parameter corresponds to the document title. The optional doc-sym parameter defaults to the value of OA$CURDOC when omitted. The value of doc-sym must correspond to a full 36-character FOLDER/DOCNUM value. The optional target-sym parameter, when

present, is an ALL-IN-1 symbol that receives the value of the next document. (The value of target-sym remains unchanged if NEXT_TITLE does not find a next document.)

Also, when NEXT_TITLE finds the next document title, both doc-sym and target-sym contain the same value, and you can use either one as the doc-sym in a subsequent call to NEXT_TITLE. That is, once the Cabinet gives these parameters a value, you can use them to maintain the stream established for NEXT_TITLE.

Assume the File Cabinet contains the following three folders and five documents:

Folder	*Number*	*Title*
GENERAL	1	Green1
GENERAL	4	Blue
GENERAL	7	Pink
PROGRAMS	2	Green2
PERSONAL	8	Green1
PERSONAL	9	Blue

The following example illustrates the results of calling CABINET NEXT_TITLE with different title-sym values:

	doc-sym value			target-sym value		
Title-sym	*Before call:*		*After call:*		*After call:*	
Green1	—	—	GENERAL	1	GENERAL	1
Green1	GENERAL	1	PERSONAL	8	PERSONAL	8
Green1	PERSONAL	8	—	—	PERSONAL	8*
Green2	PROGRAMS	2	—	—	PERSONAL	8*
Pink	—	—	GENERAL	7	GENERAL	7
Blue	GENERAL	7	PERSONAL	9	PERSONAL	9**
Green1	PERSONAL	9	—	—	PERSONAL	9*
Green1	—	—	GENERAL	1	GENERAL	1
Orange	GENERAL	1	—	—	GENERAL	1*
Orange	—	—	—	—	GENERAL	1*

* Target-sym remains unchanged; informational message is issued.
** Second occurrence of Blue is found because the current document, "GENERAL 7" comes after the first occurrence of Blue, which is with document "GENERAL 4".

OPEN

The OPEN subfunction opens the File Cabinet data set. If the File Cabinet is already open, CABINET OPEN takes no action.

All CABINET subfunctions that need access to the data set files automatically call OPEN, so you generally do not have to call CABINET OPEN explicitly.

CABINET OPEN takes no parameters.

POP

CABINET POP pops the document that was pushed onto an internally maintained stack by a preceding CABINET PUSH call. The document becomes the current document; that is, POP loads its FOLDER/DOCNUM into OA$CURDOC.

CABINET POP takes no parameters.

PUSH

CABINET PUSH pushes the current document (OA$CURDOC) onto an internally maintained stack. Call CABINET POP to pop the document off of the stack.

CABINET PUSH takes no parameters.

PUT_PENDING

CABINET PUT_PENDING updates the system pending file for the specified user. The syntax is:

```
CABINET PUT_PENDING [doc-sym], appl-sym, username-sym
```

where

doc-sym	is an ALL-IN-1 symbol representing the FOLDER/DOCNUM combination that is stored in the pending file.

appl-sym — is an ALL-IN-1 symbol representing the application (for example, MAIL) under which the information is stored.

username-sym — is an ALL-IN-1 symbol representing the name of the individual (and the key to the PENDING.DAT record) where the PUT_PENDING entry is made.

REFILE_DOCUMENT

CABINET REFILE_DOCUMENT moves an existing document to a new folder. The document is deleted from the original folder unless a cross-filed version was found in the target folder. The document retains the same DOCNUM. The syntax is:

```
CABINET REFILE_DOCUMENT [doc-sym], folder-sym [,target-sym] [,title-sym]
```

where

doc-sym — is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to refile. When omitted, it defaults to the value of OA$CURDOC.

fldr-sym — is an ALL-IN-1 symbol representing the folder in which the document is refiled.

trgt-sym — an optional parameter, is an ALL-IN-1 symbol to receive the FOLDER/DOCNUM of the refiled document. When target-sym is specified, the new refiled reference to the document becomes the current document: that is, the new FOLDER/DOCNUM is assigned to OA$CURDOC as well as to target-sym.

titl-sym — an optional parameter, is an ALL-IN-1 symbol containing the new title of the refiled document. When you omit title-sym, the document retains its original title.

NOTE: If the document already exists in the receiving folder (the result of a CROSSFILE operation), then REFILE_DOCUMENT does nothing, and the document remains in its current folder. The DELETE_OR_REFILE subfunction ensures that a document is removed from its current folder.

REORG__DOCDB

CABINET REORG__DOCDB reorganizes OAUSER:DOCDB.DAT and OAUSER:DAF.DAT to make these files more compact and more efficient to access. Call this subfunction after numerous additions to and deletions from the File Cabinet have left a user's files inefficiently fragmented.

REORG__DOCDB takes no parameters.

RESET__DELETE

CABINET RESET__DELETE resets the Delete field in the document to the specified value. The Delete field determines, from an application's point of view, if the document may be deleted. The syntax is:

```
CABINET RESET__DELETE [doc-sym], value-sym
```

where

doc-sym — is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to be changed. When omitted, this parameter defaults to the value of OA$CURDOC.

value-sym — is an ALL-IN-1 symbol representing the new value for the Delete field. The value is either Y or N.

SCAN

CABINET SCAN uses the Boolean expression supplied as a parameter to set up selection criteria for the CABINET NEXT__SCAN subfunction. The syntax is:

```
CABINET SCAN expression =
```

where expression is any valid combination of File Cabinet fields, other ALL-IN-1 symbols, and relational operators. Boolean expression syntax is covered in detail under the FOR function.

NOTE: There must be a blank space immediately preceding and following the equals sign, and the symbols must be in capitals.

For example:

```
CABINET SCAN .FOLDER = "PROJECT " AND .FORMAT = = "RUNOFF"
```

scans the File Cabinet and establishes a context for NEXT_SCAN that includes all documents stored in folders beginning with PROJECT that have formats equal to RUNOFF. SCAN only sets the context: CABINET NEXT_SCAN uses the SCAN-supplied criteria to actually select a next document.

SELECT

The SELECT subfunction makes a File Cabinet document current and stores its FOLDER/DOCNUM in special symbol OA$CURDOC. CABINET SELECT is similar to CABINET CURRENT, but SELECT accepts more parameters, and thus more selection criteria.

The syntax is:

```
CABINET SELECT [folder-sym], [number-sym], [title-sym] [,target-sym]
```

where

folder-sym — an optional parameter, is an ALL-IN-1 symbol representing the FOLDER of the document to select. When you omit this parameter, the selection criteria default to the first document matching the other parameters.

number-sym — an optional parameter, is an ALL-IN-1 symbol representing the DOCNUM, the system-assigned document number, of the document to select. When you omit this parameter, the selection criteria defaults to the first document matching the other parameters.

title-sym — an optional parameter,is an ALL-IN-1 symbol representing the title of the document to select. When you omit this parameter, the selection criteria default to the first document matching the other parameters.

target-sym — an optional parameter, is an ALL-IN-1 symbol to receive the FOLDER/DOCNUM of the selected document. OA$CURDOC also contains the FOLDER/DOCNUM of the selected document.

The following example illustrates how CABINET SELECT determines the current document based on the contents of the parameters. An X indicates that the parameter has some value, while "–" indicates that it is null.

Folder	Number	Title	Key Used *	Action
X	X	X	0	Exact document only if title matches
X	X	—	0	Exact document
X	—	X	0	First document in specified folder with matching title
X	—	—	0	First document in specified folder
—	X	X	1	First occurrence of document number with matching title
—	X	—	1	First occurrence of document number
—	—	X	2	First occurrence of document with matching title
—	—	—	0	First document in File Cabinet

* KEY specifies the RMS key of reference used.

SHARE_DOCUMENT

CABINET SHARE_DOCUMENT moves the RMS file associated with a private document to the shared area and moves that document's attribute record from the PDAF to the SDAF. SHARE_DOCUMENT adds some DOCDB.DAT fields to the SDAF. If a document is already shared, SHARE_DOCUMENT does nothing.

The syntax is:

```
CABINET SHARE_DOCUMENT [doc-sym]
```

where doc-sym is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to share. This parameter is optional and defaults to OA$CURDOC if omitted.

SHARE_DOCUMENT uses logical OA$SHARE_CUR to determine where in the shared area to store the document. This subfunction also:

- Moves attachments to the document that were private to the shared area
- Increments the usage counts of attachments that are already shared

For example:

CABINET SHARE $MAILDOC

STOP_SCAN

CABINET STOP_SCAN stops a scan started by CABINET SCAN.

The syntax is:

CABINET STOP_SCAN

UNSHARE_DOCUMENT

CABINET UNSHARE_DOCUMENT does the following:

- Removes a document from the shared area and updates the File Cabinet records associated with this file to indicate that it is no longer shared.
- Makes a copy of the document in the user's private File Cabinet.
- Copies all the attributes into the user's PDAF.
- Decrements the usage count of the original shared document; if the usage count is zero, CABINET UNSHARE_DOCUMENT deletes the shared copy.

The syntax is:

CABINET UNSHARE_DOCUMENT [doc-sym]

where doc-sym is an ALL-IN-1 symbol representing the FOLDER/DOCNUM of the document to unshare. This parameter is optional and defaults to OA$CURDOC if omitted.

10

Text Processing

10.1 Overview

Text processing is creating, editing, combining, and outputting material to be read. It covers graphics as well as words and list processing functions as well as interactive editing.

How a file is created and stored in ALL-IN-1 does not necessarily limit how it can be edited and combined with other files. Each supported editor in ALL-IN-1 processes text differently. The default text editor is a callable version of an EDT-style VMS editor, and it creates and operates on standard ASCII-formatted text files. The WPS-PLUS text editor, available as an ALL-IN-1 option, creates and operates on specially formatted files that are nearly identical in appearance and behavior to WPS-8 standalone word processing documents. You can transfer to ALL-IN-1 any WPS-8 documents that you have created on standalone word processors, and then use any ALL-IN-1 function on those documents.

NOTE: A WPS-8 document is much richer in structure and data than an ASCII document. Thus, if you try to edit it with the default editor, you lose some of this structure and data. The optional WPS-PLUS editor must be used to edit WPS-8 documents without loss of structure or data.

To achieve this flexibility, ALL-IN-1 treats text files as data sets, as covered in the following section.

10.2 Text Data Sets

A text data set is one or more VMS sequential files and information about the contents of those files that ALL-IN-1 maintains while you are processing the file(s). For most interactive operations, ALL-IN-1 processes text data sets a line at a time, in the context of a standard text format.

This standard text format is the key to the text data set: it is an internal mechanism that allows ALL-IN-1 to treat all files in the same way, regardless of their on-disk structure. The text data set mechanism also allows you, for example, to create a file with any supported text editor and then convert that file into any other supported format for further editing, printing, terminal output, and so on.

ALL-IN-1 groups text data sets into the following categories:

- ASCII — ASCII-formatted files. This is the text type for files created with the default editor and the default text type for all unknown file formats. The corresponding file extension is .TXT.
- COMPOUND — Compound documents, which are documents that are comprised of two or more documents, which may be any other type. The MAIL document type is a special case of compound document where a MAIL_HEAD document must be the first document in the compound document, and where there may be only one MAIL_HEAD document present in the compound document.
- DX — DECdx-formatted files. These are files that have been transferred from WPS-8 standalone word processors. The corresponding file extension is .DX.
- DXT — transfer of DECdx-formatted files between ALL-IN-1 and a DECmate II personal computer.
- MAIL_ATTACHMENT — Mail attachments, used in the Electronic Messages subsystem.
- MAIL_HEAD — Mail header "FILES", used in the Electronic Messages subsystem. These "FILES" are maintained as data in the user's file cabinet database and do not exist as actual text files.

- PIPELINE — Pipeline output.
- WPSPLUS — WPS-PLUS-formatted files, created with the optional WPS-PLUS editor. The corresponding file extension is .WPL.

10.3 Creating Files with the EDIT Function

When you create a file with the EDIT function, ALL-IN-1 gives the document the format specified in the User Profile Editor Style field, EDITOR, unless you explicitly provide an alternate in the EDIT function call. ALL-IN-1 recognizes the editor styles listed in Table 10.1.

Table 10-1 Recognized Editor Styles

Style	Internal Structure	VMS File Extension (OA$DEFAULT_EXTENSION)
EDT	ASCII	.TXT
WPS	ASCII	.TXT
DECTEXT	DECTEXT	.DOC
DECTYPE	DECTYPE	.DTP
DX	DECdx	.DX
DX	Extended 8-bit	.DX8
WPSPLUS	WPS-PLUS	.WPL
other	ASCII	.TXT

NOTE: The EDT and WPS editor styles are the two variations of the default ALL-IN-1 editor. All other editors are optional and may not be available on your system.

ALL-IN-1 recognizes other file-type extensions that correspond to more general file processors as opposed to text editors.

As illustrated, ALL-IN-1 associates each editor style internally with a VMS file-type extension. If the contents of User Profile field EDIT correspond to a supported editor style, ALL-IN-1 loads OA$DEFAULT_EXTENSION, a special symbol with the value of the corresponding default extension. When ALL-IN-1 invokes a text editor on a document, it first goes to the file cabinet and looks up the document's editor style. If the document does not exist, then ALL-IN-1 uses the file-type extension to decide how to process

the text file. You can use the extension, through the /DOC field qualifier and MAKE_FILE_NAME function, to build a unique file name for newly created files.

10.4 Features of the Default Editor

The ALL-IN-1 default editor is a callable version of an EDT-style text editor. Files you create with this editor are stored in ASCII format and are associated with the file-type extension .TXT. They may have either the EDT or WPS edit styles associated with them.

When you call the EDIT function, EDIT looks to the value of updatable special symbol OA$EDITOR to decide which editor to call.

If OA$EDITOR has the value EDT or WPS, the EDIT function invokes the default editor.

Like EDT, the default ALL-IN-1 editor runs a start-up file to associate key sequences and macros with editing tasks. When you specify the EDT editor, EDIT calls the default editor with one set of start-up definitions; and when you specify the WPS editor, EDIT calls the default editor with another set of definitions. These start-up definitions are the basic differences between the two default editors.

The EDT and WPS start-up files are coded internally for performance reasons, but you can specify your own start-up file as the second EDIT parameter. Any definitions in your start-up file are added to the internal definitions.

The *User's Reference* covers other features of the editor in detail; the following subsections describe how some of those features are implemented.

10.4.1 Editor Modes

- Line Mode — You enter Line Mode commands at the asterisk prompt on line 24 to modify text at the line level.
- Change Mode — You press keypad keys, Gold-key sequences, and/or CTRL/key sequences that call nokeypad commands that allow you to move around on the screen and modify text at the character level.

Both WPS and EDT initialize in Change Mode, but you can enter Line Mode at any time by pressing Gold Z or CTRL/Z. You can define keys in Change or Line Mode, and you define macros in Line Mode.

10.4.1.1 Nokeypad Commands, Buffers, Key Definitions, and Macros.

A nokeypad command is an abbreviated word that, when invoked, performs an editing task. You invoke nokeypad commands in Change Mode, either by pressing the associated keys as covered previously; or by entering the nokeypad command directly in Nokeypad Mode, which is covered in the EDT documentation set.

A buffer is a workspace that holds text. Outside of ALL-IN-1, the editor initializes by default with the following:

- MAIN — The main edit buffer, where the body of the document is stored while you edit it.
- PASTE — The paste buffer, which holds text temporarily during the edit session for cut and paste operations.

You can create any number of other buffers during the edit session to store text or editing commands. Several commands that require buffers automatically create these buffers if they do not already exist; to create a buffer interactively in Line Mode, enter one of the following commands at the asterisk prompt:

```
* =buffer_name
```

OR

```
* BUFFER buffer_name
```

where the = identifies what follows as a buffer name.

A macro is an edit buffer that contains Line Mode commands. You define a macro with the DEFINE MACRO command:

```
DEFINE MACRO macro_name
```

where macro_name is a name that you want to associate with the macro buffer. You can associate a macro with a key sequence so that the editor executes those commands when you press the key sequence. The default editor creates several macros when it initializes. You create others either at start up or during the session.

Associate key sequences with nokeypad commands in the following ways:

- With the Change Mode DEFK command, which is associated, by default, with CTRL/K. When you invoke DEFK, the editor prompts you to first press the key sequence you wish to define, and then records your keystrokes and associates them with the key.

  As covered in the following paragraphs, the editor interprets your key definition as a series of nokeypad commands and executes those commands when you press the defined key. As with DEFINE KEY, you should terminate the command string with a period if you want the editor to execute the commands immediately.

- With the Line Mode DEFINE KEY command. The syntax is:

  ```
  DEFINE KEY key-name AS 'definition string'
  ```

  where

  key-name is one or more keywords that identify the key sequence. The valid key names are:

  - [GOLD] [FUNCTION] number — DEFINE KEY number refers to keypad keys, DEFINE KEY GOLD number refers to Gold-keypad sequences, DEFINE KEY FUNCTION number refers to top-row function keys on the LK201 keyboard, and DEFINE KEY GOLD FUNCTION number refers to Gold function-key sequences.
  - [GOLD] CONTROL character — DEFINE KEY CONTROL character refers to CTRL/key sequences, and DEFINE KEY GOLD CONTROL character refers to Gold CTRL/key sequences. The keys you can define are @, A to Z, [,], \, ^, and _.
  - [GOLD] DELETE — DEFINE KEY DELETE and DEFINE KEY GOLD DELETE refer to the delete-previous-charater (DEL) or RUB CHAR OUT keys on the keyboard.
  - Gold character — DEFINE KEY GOLD character refers to Gold main keyboard-key sequences. You can define any key except 0 through 9.

    NOTE: *Enclose the special characters !, %, ', and " in quotes or apostrophes when defining them with the above sequences.*

'definition string'	is a string of nokeypad commands that the editor executes when you press the defined key. You must enclose the entire string either in quotes or apostrophes, and typically you end the string with a period before you close the quote. The editor then executes the command immediately; otherwise, commands accumulate and are executed when the editor encounters the first command that ends with a period.

The DEFINE KEY string, except for the quotes, corresponds to the keystrokes you enter when prompted by DEFK. For example:

```
DEFINE KEY GOLD 1 AS 'IPress GOLD 1 to insert this text^Z.'
```

This command redefines the sequence Gold keypad 1 to execute the nokeypad INSERT command (abbreviated 1). The editor inserts the specified text, up to ^Z, into the text at the current cursor position.

```
DEFINE KEY CONTROL" ' " AS "ADVI^*^Z?*'Bold?:'W-1CI\*^Z."
```

When you press CTRL/', the editor does the following:

1. Sets the cursor direction to ADVANCE.
2. Inserts the characters "^*" in text at the current cursor position.
3. Prompts the user to enter a number by displaying "Bold?:" on line 24 and waiting for a response. The prefix "?*" indicates a prompt; you must enclose the text of the prompt in apostrophes. (You can also prefix the prompt with just "?", but this limits the range of terminator keys that the user can press to enter the answer.)
4. Moves forward X words, where X is the number entered by the user at the prompt; moves back one character; and inserts the characters "*".

```
DEFINE KEY GOLD C as "(ADVLI.!^Z)."
```

When you enclose part or all of a key definition in parentheses, you can use repeat counts with the key. Thus pressing Gold C when it has been defined as here advances one line and inserts the characters ".!"; while the following key sequence (from Change Mode) inserts the characters ".!" in front of the next 600 lines:

{GOLD 6)00(GOLD C}

The preceding example denotes the way user keyboard responses are represented in ALL-IN-1 script files. They appear at various times in this APR to show what the user is being asked to type. See Chapter 15 for more information on scripts.

To associate a key sequence with a macro, use the EXTEND nokeypad command (abbreviated "EXT"). For example:

```
DEFINE KEY GOLD _ AS "EXT UNDERLINE."
```

The EXTEND command instructs the editor to treat the next command as a Line Mode command. Since you can enter macro names while editing in Line Mode to execute commands in a macro, you can invoke a macro in Change Mode by associating a key sequence (in this case, Gold_) with EXTEND macro-name. (You must define UNDERLINE as a macro with the DEFINE MACRO command before you press Gold _.)

You generally load a macro with Line Mode commands from within an initialization file, as covered after the following list.

You can define key sequences with DEFK interactively while editing, and you can define key sequences with DEFINE KEY as follows:

- In an initialization file. EDIT processes all initialization file entries as Line Mode commands before you enter the editing session. Just begin each Line Mode command on a new line, or separate commands from one another with semicolons. For example:

  ```
  DEFINE KEY GOLD 1 AS "IYou just pressed GOLD 1Z."
  DEFINE KEY GOLD 2 AS "IYou just pressed GOLD 2Z."
  ```

- In Line Mode, while editing. Press Gold Z or CRL/Z while in Change Mode, then enter the DEFINE KEY command at the asterisk prompt. Press RETURN or ENTER to end the command, and enter the Line Mode command CHANGE (abbreviated C) to return to Change Mode. For example, the following sequence begins while Change Mode is the Current Mode:

  ```
  [GOLD Z]
  *DEFINE KEY GOLD 3 as "IYou just pressed GOLD 3^Z."{RETURN}
  *C
  ```

  Keypad sequence Gold 3 now inserts the specified text when pressed.

The following example illustrates how you can define a macro in an initialization file and then load it with Line Mode commands using the INSERT command. The example shows how ALL-IN-1 defines the key sequence Gold # so that it displays the desk calculator form; the lines that begin with an exclamation point are comments that have been added for clarity:

```
!
!                   Define a macro to call the desk calculator.
!
DEFINE MACRO CALC
!
!                   Make the macro buffer the current buffer, and insert
!                   Line Mode commands in it.
INSERT = CALC
!
!                   In the CALC buffer now. . . . First clear the PASTE
!                   buffer, then make it the current buffer during the
!                   operation so that the user can include all results
!                   into the document.
!
CLEAR PASTE
FIND  = PASTE
!                   In the   PASTE buffer now. . . Set Mode to CHANGE so that
!                   we can   call some nokeypad commands:
!                                XLATE    returns control to ALL-IN-1: ALL-IN-1
!                                         processes everything up to the ^Z as
!                                         a function call. Form OA$DESK
!                                         displays.
!                                REF      refreshes the screen.
!                                EX       exits from CHANGE Mode to LINE mode.
!                                FIND     is a Line Mode command that returns
!                                         from buffer PASTE to buffer CALC.
!
CHANGE;XLATE OA FORM OA$DESK\CLOSE_PRIOR^ZREFEXFIND .
!
!                   Done with the CALC form. Return to the last buffer
!                   (usually MAIN).
FIND LAST
!                   ^Z indicates the end of the INSERT input and thus
!                   the end of buffer CALC.
^Z
```

10.4.2 Communicating with the Rest of ALL-IN-1 Through the Editor

10.4.2.1 The XLATE Command. The XLATE nokeypad command in the preceding macro is a feature of the default editor that allows you to call ALL-IN-1 functions while editing. The syntax is:

```
XLATE OA function[\function...] ^Z
```

where everything between XLATE and ^Z is passed to ALL-IN-1 as a function call. The OA prefix is required syntax.

Since XLATE is a nokeypad command, you can associate it directly with a key sequence using a single-line DEFINE KEY command. However, to ensure that the XLATE/ALL-IN-1 interaction is consistent throughout the editor, you usually should call XLATE from within a macro, as illustrated in the previous section.

10.4.2.2 The DESTINATION Function and OA$EDIT_BUFFER. While you are in the editor, the OA$EDIT_BUFFER text data set is open and current. When you call a function through XLATE, ALL-IN-1 automatically precedes the call with a call to the DESTINATION function: that is, when you specify:

```
XLATE OA function[\function...] ^Z
```

ALL-IN-1 automatically dispatches the function call:

```
DESTINATION OA$EDIT_BUFFER\function[\function...]
```

DESTINATION sends the results of any output-generating function (GET, COMPUTE, DTR, VERSION, and so forth) to the specified data set, in this case OA$EDIT_BUFFER, which is the current edit buffer. Thus, when you press Gold # while editing and the editor invokes a macro CALC (the example macro illustrated previously), the following happens:

1. Gold # invokes the Line Mode commands in macro CALC. The first two commands clear the paste buffer and make it the current buffer.
2. The next command switches back temporarily to Change Mode to execute XLATE and some other nokeypad commands.

The editor passes the XLATE string to ALL-IN-1 as:

```
OA FORM OA$DESK\CLOSE_PRIOR
```

ALL-IN-1 in turn dispatches the call:

```
OA DESTINATION OA$DIT_BUFFER\FORM OA$DESK\CLOSE_PRIOR
```

3 Form OA$DESK displays. This form calls the COMPUTE function internally to evaluate expressions. Thus, any answers that display on the form are generated by COMPUTE, which also sends the answer back to OA$EDIT_BUFFER.

   The call to CLOSE_PRIOR ensures that subsequent calls to forms from within the editor do not confuse EXIT SCREEN processing.

4 The remaining commands clean up after the XLATE and ensure that you return to the buffer and line at which you pressed Gold #. Anything that the DESTINATION function has channeled to OA$EDIT_BUFFER is now in the PASTE buffer.

5 If you press the PASTE key, the results, if any, which you generated while using the desk calculator are pasted into text.

A similar process takes place when you call an output-generating function interactively from the choice field of the Editor menu or any other menu that you call through XLATE. ALL-IN-1 stores the results of that function call in the buffer from which you call XLATE. When you return to the current edit buffer, you can include those results with one key sequence.

The following sections cover commonly used nokeypad and Line Mode commands used during editing. Each editing key sequence is associated with a nokeypad command string or with a macro that contains Line Mode commands. To view these key definitions, use the Line Mode command SHOW KEY key-name, where key-name corresponds to the number or character that you specify when defining the key.

Figure 10-1 illustrates the numbers of keys that are not obvious. Use these numbers with the SHOW KEY and DEFINE KEY commands.

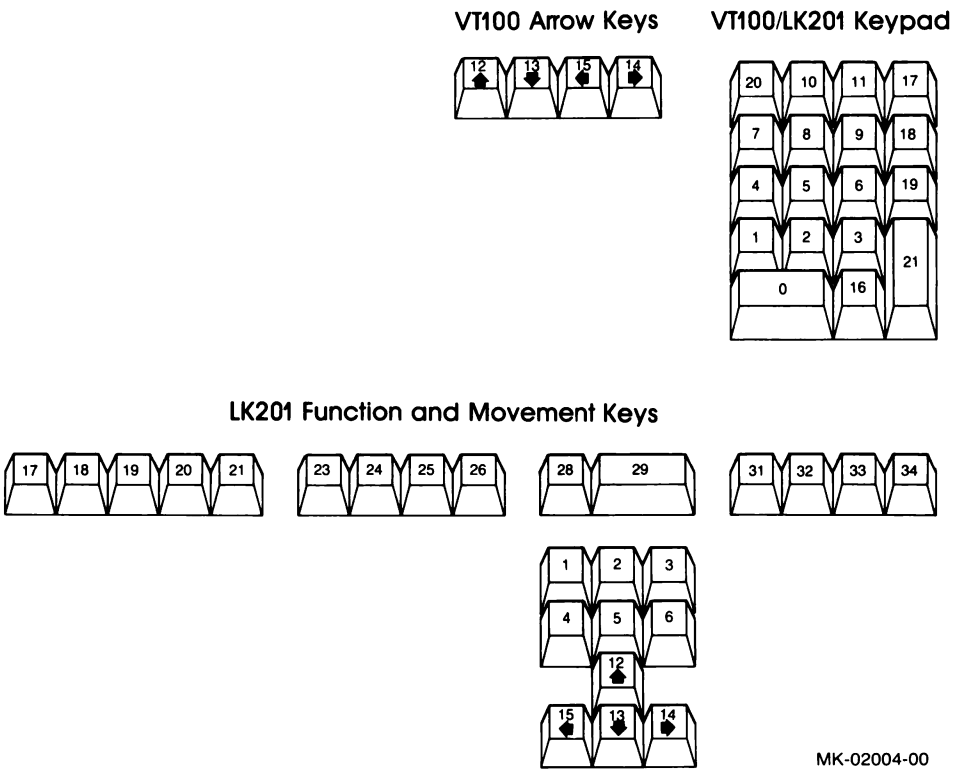


Figure 10-1 Key Numbers for the Default Editor

As mentioned earlier, when you call the default editor through the EDIT function, ALL-IN-1 determines whether you want to invoke the EDT variation or the WPS variation. The only difference between the two editors is which keys are associated with which nokeypad commands and macros. You can always determine what commands are associated with a given key with the SHOW KEY command.

10.4.3 Moving Within a Document

There are many commands that allow you to move within a document; the following list is a brief overview of the most common command sequences.

- Advancing through the document

  To set the current cursor direction to ADVANCE, invoke the nokeypad ADV command. The following descriptions assume that the current cursor direction is ADVANCE.

  - To move one character, invoke the +C or ADVC nokeypad sequences.
  - To move one word, invoke the +W or ADVW nokeypad sequences.
  - To move to the end of the current line, invoke the +EL or ADVEL nokeypad sequences.
  - To move to the beginning of the next line, invoke the +L or ADVL nokeypad sequences.
  - To move directly down one line, press the Down Arrow key or the key that invokes the +V nokeypad command.
  - To advance 16 lines while in EDT, press keypad 8. To advance one paragraph while in WPS, press keypad 5.
  - To move to the bottom of the document, invoke the nokeypad sequence ER (end of range).

- Moving back through the document

  To set the current cursor direction to BACK, invoke the nokeypad BACK command. The following descriptions assume that the current cursor direction is BACK.

  - To move one character, invoke the -C or BACKC nokeypad sequences.
  - To move one word, invoke the -W or BACKW nokeypad sequences.

  - To move to the end of the previous line, invoke the -EL or BACKEL nokeypad sequences.
  - To move to the beginning of the previous line, invoke the -L or BACKL nokeypad sequences.
  - To move directly up one line, press the Up Arrow key or the key that invokes the -V nokeypad command.
  - To back up 16 lines while in EDT, press keypad 8. To back up one paragraph while in WPS, press keypad 5.
  - To move to the top of the document, invoke the nokeypad command BR (beginning of range).

- Moving to a specified point

  To move to a specified point in the text (that is, to search for a specified string), first set the direction to BACK or ADVANCE. Then:

  - In Change Mode, press Gold PF3 or the key sequence that invokes a nokeypad sequence that prompts you for the string to search for and then searches for it.
  - In Line Mode, enter the string at the asterisk prompt, enclosed in quotes.

- Moving to a specified line

  To move to a specified line in the text from Line Mode, just enter the line number at the asterisk prompt.

10.4.4 Moving Text Within a Document

The simplest way to move text around in a document is through the CUTSR (cut select range and transfer it to the paste buffer) and PASTE (include contents of paste buffer in text) nokeypad commands. The CUTSR command is invoked by the keypad comma in WPS and by keypad 6 in EDT. The PASTE command key is invoked by the keypad HYPHEN (-) in WPS and by Gold keypad 6 in EDT.

Before you invoke CUTSR, you must select a range of text with the SEL nokeypad command. In both default editor styles, the keypad PERIOD (.) invokes the SEL command. After you invoke this command, you move to some other point in the text, and all intervening text becomes the select range. To cancel a select range, invoke the RESET nokeypad command (Gold keypad period in both editor styles).

The PASTE nokeypad command is a convenient way to include the contents of the paste buffer into text at the current cursor position. Since the paste buffer is, in many ways, just another edit buffer, you can include text by using other commands besides CUTSR. For example, the APPENDSR nokeypad command appends the current select range to the current contents of the paste buffer; CUTSR simply replaces the contents. In the EDT editor, keypad 9 invokes APPENDSR. Use APPENDSR when you want to accumulate additions to the paste buffer.

Also, several of the macros defined during editor initialization leave information in the paste buffer, allowing you to include this information readily into the main body of the document.

The COPY Line Mode command is another way to move text within a document. The general syntax is:

```
COPY [range-1] TO [range-2] [/QUERY] [/DUPLICATE:n]
```

where

range-1	identifies the text to copy; when omitted, this range defaults to the current line.
range-2	is where range-1 is copied to; when omitted, this range defaults to the current line.
/QUERY	is an optional qualifier that, when specified, prompts you with a question mark to each line copied.
/DUPLICATE:n	is an optional qualifier (abbreviated /DUP) that causes the lines in range 1 to be duplicated n times.

You can specify a range in many ways: as a buffer, a beginning/ending line-number combination, and so on. For example:

```
COPY =TEMP TO
```

copies the contents of buffer TEMP into the current buffer, starting at the current line.

```
COPY SEL TO =TEMP END
```

copies the current select range into buffer TEMP, appending it at the end. (The default action is to copy text to the beginning of a buffer.) In all cases, COPY does leave the text in range 1 intact.

There are many other ways to specify ranges; see the EDT documentation set for a full description of this concept.

10.4.5 Including Outside Text into a Document

When the default editor initializes, several key sequences are defined to allow you to readily include outside text into the document. Outside text refers to VMS files, VMS files indexed through the File Cabinet, and the results of certain function calls.

To include a VMS file into the current buffer, press CTRL/F, which is associated with the nokeypad sequence:

```
EXT INCLUDE ?*'File Name?: '."
```

This sequence prompts for a VMS file, and then calls the Line Mode INCLUDE command to include that file into text at the current cursor position. You can call INCLUDE (abbreviated INC) in Line Mode to do the same thing:

```
* INC file_spec
```

where file_spec is any valid VMS file specification. You can specify node names and include files that are on another machine.

ALL-IN-1 also has an INCLUDE function that allows you to include File Cabinet documents into the current edit buffer. The INCLUDE syntax is:

```
INCLUDE form_name, file_field
```

where form_name is a form that displays and prompts for a File-Cabinet document name, and file_field is the field on that form that contains the VMS file name of that document.

NOTE: The ALL-IN-1 INCLUDE function is available in ALL-IN-1 Version 2.0 for compatibility with Version 1. The preferred method for including document text is the GET macro, explained next.

In both the EDT and WPS styles, Gold G invokes macro GET, which contains the following single line:

```
CHANGE;XLATE OA FORM EDTGMEN\CLOSE_PRIOR^ZREFEXFIND .
```

When you press Gold G:

1. The editor enters Change Mode.
2. ALL-IN-1 displays form EDTGMEN. The user specifies the document to be selected via this form.
3. ALL-IN-1 sends the specified document to the current edit buffer.
4. ALL-IN-1 processes the CLOSE_PRIOR call.
5. The editor processes the cleanup commands.

As discussed previously, the DESTINATION function lets you include concise information — function results — into the current edit buffer.

While you are editing you can use the DCL function to enter the subprocess to create files, by entering a VMS editing session. When you return to the ALL-IN-1 editor, you can then include that file or files into text.

The predefined macro DCL calls the DCL function; thus, entering DCL at the Line Mode prompt puts you in the subprocess at DCL level. Also, Gold $ invokes this macro.

10.4.6 Creating New Files While in the Editor

As just mentioned, you can call the DCL function to enter DCL level while editing. At this level, you can use all the tools of DCL to create and/or modify files.

Also, the Line Mode WRITE command lets you create files while editing. The abbreviated syntax is:

```
* WRITE file_spec [range]
```

where file_spec can be any valid VMS file specification, including a node name; and range identifies the lines to write to that file. When you omit the range, it defaults to the entire current buffer.

10.4.7 Customizing the Default Editor

The preceding sections are just an overview of the capabilities of the default editor. For a more detailed explanation, see the EDT documentation set.

When defining keys or using Line Mode commands, either within the editor or in an initialization file, you may confuse Line Mode commands and nokeypad commands. A list of facts to remember when customizing the editor follows.

- ALL-IN-1's default editor is automatically in Change Mode if the user's terminal has a screen, and in Line Mode if the terminal is a hard-copy terminal. To reenter Change Mode from Line Mode, enter the CHANGE command (abbreviated C) at the asterisk prompt.
- You enter Line Mode interactively by pressing Gold Z or CTRL/Z. Enter multiple Line Mode commands by separating each command with a semicolon.
- The editor recognizes Line Mode commands in Line Mode and nokeypad commands in Change Mode. (These two different sets of commands sometimes overlap; for example, to search for a string, you enclose the string in quotes in both Line Mode and in Nokeypad Mode.)
- The editor processes initialization files as a series of Line Mode commands, unless the line has been entered in the context of some other command, such as INSERT. You are then automatically placed in Change Mode, regardless of whether you have specified CHANGE as the last command in the start-up file.
- DEFINE KEY is a Line Mode command that allows you to associate a series of nokeypad commands with a key. Use it either at the Line Mode asterisk or within an initialization file.

  To return temporarily to Line Mode at the end of the Change Mode nokeypad sequence, use the nokeypad command EXTEND (or EXT). The editor processes everything following the EXT in Line Mode. You can only use EXTEND as the last command in a nokeypad sequence, and you cannot use extend within parentheses. After the editor processes the Line Mode command, it automatically returns to Change Mode.

- To enter Line Mode from Change Mode and stay in Line Mode, use the EXIT (or EX) nokeypad command. This EXIT command is different from the EXIT Line Mode command, which exits the editor.
- DEFINE MACRO is a Line Mode command that allows you to define an edit buffer as a macro. The macro itself contains Line Mode commands.

  To invoke a macro from Line Mode, just enter the macro name at the asterisk prompt. To invoke a macro from Change Mode, use:

  ```
  DEFINE KEY key-name AS "EXT macro-name."
  ```

  To return temporarily to Change Mode while in Line Mode in a macro or initialization file, use the Line Mode command CHANGE, followed by a list of nokeypad commands. The syntax is:

  ```
  CHANGE [range] [;nokeypad commands]
  ```

  where range specifies the cursor position when you enter Change Mode and defaults to the current cursor position when you omit it. Separate the CHANGE command from the nokeypad commands with a semicolon.

  To return to Line Mode in this context, include the EXIT nokeypad command at the end of the nokeypad sequence. This command was covered previously.

For example, you can create files while editing by calling the DCL function and then invoking DCL commands that generate output. To simplify the inclusion of these files into the editor, create the following initialization file and call it DCL.EDT:

```
!+
! Associate logical PASTE with temporary file PASTE.TMP...
!-
CHANGE;XLATE OA GET OA$DCL = "DEF/PROC PASTE PASTE.TMP"^ZEX
!+
! Put an empty PASTE.TMP in OA$USER to avoid error messages
! later during the INCLUDE.
!-
WRITE PASTE.TMP  =DUMMY
!+
! Clear the DCL buffer of its predefined contents
!-
CLEAR DCL
INSERT = DCL
CLEAR PASTE
FIND = PASTE
CHANGE;XLATE OA DCL\CLOSE_PRIOR^ZREFEXFIND
!+
! INCLUDE PASTE.TMP puts any output into the paste buffer
!-
INCLUDE PASTE.TMP
!+
! Delete PASTE.TMP
!-
CHANGE;XLATE OA GET OA$DCL = "DELETE PASTE.TMP;*"^ZEX
FIND LAST
^Z
FIND = MAIN
!+
! Define the macro now
!-
DEFINE MACRO DCL
```

When you call the editor as follows:

```
EDIT 'file.ext','DCL.EDT'
```

EDIT creates macro DCL as defined in file DCL.EDT. When you press Gold $ (Change Mode) or enter DCL (Line Mode) while editing, you are placed at the $ prompt in the subprocess. Any output that you direct to PASTE is stored in file PASTE.TMP and can be recalled when you return to the editor by pressing the PASTE key. For example, starting in Line Mode:

```
* DCL{RETURN}
$ DIR/OUT=PASTE/COLUMN=1 *.TXT{RETURN}
$ {RETURN}
```

Pressing the PASTE key now includes a listing of all .TXT files into the document.

10.5 Features of the WPS-PLUS Editor

The WPS-PLUS/ALL-IN-1 documentation set covers this optional editor in detail. See also the MERGE_LIST, SORT_LIST, and TEST_SPEC functions.

NOTE: Both the default editor and the WPS-PLUS optional editor convert files to the proper internal style before putting the user in the editor. Both editors then convert the edited file back into the original style when the user finishes the edit session.

10.6 List Processing

ALL-IN-1 provides two separate list-processing facilities — WPS-PLUS-based list processing through the WPS-PLUS List Processing feature and ALL-IN-1 list processing through the interaction of the MERGE, MERGE_LINE, and OA$MRG_TTOUT functions. The WPS-PLUS List Processing feature, which is only available when WPS-PLUS is installed with ALL-IN-1, is implemented through the MERGE_LIST, SORT_LIST, and TEST_SPEC functions. Refer to these functions and to the WPS-PLUS/ALL-IN-1 documentation set for information on how to use this feature.

The MERGE and MERGE_LINE functions provide similar list-processing capalities. The following list briefly summarizes MERGE features.

- MERGE processes templates that contain directives. The directives, enclosed in angle brackets, can be symbols, function calls, and/or special MERGE directives.

- Any text file can serve as an optional MERGE list file. Use the list file when you want MERGE to process the template repeatedly. In a template, refer optionally to the current list-file record through the OA$MERGE_KEY special symbol.
- To create multipage reports, just enter a form-feed or a directive that causes a page break in the template. To number pages and/or add running header information, create special header/footer templates and specify them as MERGE parameters.
- Call MERGE_LINE and OA$MRG_TTOUT interactively to test MERGE directives.

For example, template DOHELP.BLP could contain:

```
1 <$FRM_NAME>       <&OA .IF #FIELD_NAME = = "CHOICE" <->
                    THEN .FX GET #TYPE = "Menu" <->
                    ELSE .FX GET #TYPE = "Form" >
.BR; The xxx <#TYPE>
.PS,75              <&OA .IF #FIELD_NAME = = "CHOICE" <->
                    THEN .FX GET #L2 = "2 " OA$MENU_CHOICE <->
                    ELSE .FX GET #L2 = "2 " #FIELD_NAME>
.NO PAGING
.AUTOPARAGRAPH      <&OA .IF #FIELD_NAME = = "CHOICE" <->
                    THEN .FX GET #L4 = "The xxx Option" <->
                    ELSE .FX GET #L4"The xxx Field" >
.SET PAR 0,1

<#L2>
.br
<#L4>
```

A Named Data entry, command procedure, or script loads symbols $FRM_NAME and #FIELD_NAME before MERGE processes this template. When you specify:

```
MERGE DOHELP.BLP,NEW.HLP
```

MERGE processes DOHELP.BLP left to right, top to bottom. The resulting output file, NEW.HLP, looks like:

```
1 ANYFORM
.BR;The xxx Form
.PS,75
.NO PAGING
.AUTOPARAGRAPH
.SET PAR 0,1

2 ANYFIELD
The xxx Field
```

In this case, MERGE generates a brief file with commands that, when processed through RUNOFF, yields a formatted HELP module that you can insert into OA$LIB:OAHELP.HLB; you can insert NEW.HLP itself into FAC$COMMON:HLPSOURCE.HLB.

When you want to generate a multipage report with a list file such that each list-file record ends up on its own page, simply put a <&PAGE> directive or an explicit {FF} in the template. For example:

<&PAGE>

<OA$MERGE_KEY> is the current record in the list file.

. . . . other text and directives . . . .

MERGE repeats this sequence for each line in the list file. If you also specify a header-file parameter with this type of template, MERGE processes that header-file for each record in the list file — that is, each time MERGE encounters the page marker.

MERGE automatically creates page breaks if you simply specify the header-file parameter. The following example generates a list of trees by state.

List file SPECIES.LIS contains the data that MERGE uses to create the report:

```
SPECIES—COTTONWOOD
ARIZONA                 0                       7000
COLORADO                4000                    6500
MONTANA                 2500                    5200
NEW MEXICO              4000                    7000
UTAH                    2000                    6500
SPECIES—ENGLEMANN SPRUCE
ARIZONA                 7800                    12000
COLORADO                7000                    13400
MONTANA                 6000                    12000
NEW MEXICO              7800                    13800
UTAH                    7000                    13400
SPECIES—JUNIPER
ARIZONA                 2000                    8000
COLORADO                4000                    6800
MONTANA                 2500                    6600
NEW MEXICO              4000                    8000
UTAH                    4000                    6800
SPECIES—PINON PINE
ARIZONA                 2000                    8000
COLORADO                4000                    6800
MONTANA                 2500                    6600
NEW MEXICO              4000                    8000
UTAH                    4000                    6800
SPECIES—PONDEROSA PINE
ARIZONA                 6500                    8000
COLORADO                5000                    7800
MONTANA                 4000                    7000
NEW MEXICO              6500                    8000
UTAH                    5000                    7000
SPECIES—QUAKING ASPEN
ARIZONA                 7800                    12000
COLORADO                7000                    13400
MONTANA                 6000                    12000
NEW MEXICO              7800                    13800
UTAH                    7000                    13400
```

If you specify SPECIES.LIS as a list-file parameter, MERGE associates OA$MERGE__KEY with the current line in the file; that is, OA$MERGE__KEY contains the entire line and not just the first word or token in the line.

To reference part of OA$MERGE__KEY — for example, the phrase "ENGLEMANN SPRUCE" — use the standard symbol-substring syntax:

```
OA$MERGE__KEY:16:11
```

MERGE translates all list-file TABs into spaces, so if you use the symbol-substring syntax to reference a list file that contains TABs, allow for the full spacing of each TAB character.

Generally you build a list file like SPECIES.LIS indirectly, either using the form and field qualifiers that generate text files (for example, /LIST) or using DCL file-manipulation commands such as OPEN, WRITE, and CLOSE.

Template DOTREES.BLP contains the following extended directive:

```
<&OA .IF OA$MERGE__KEY:7:0 = = "SPECIES"<->
      THEN .FX GET #SPECIES=OA$MERGE__KEY:16:11\\<->
           GET OA$PAGE__LENGTH="1"<->
      ELSE GET #A = OA$MERGE__KEY:5:32\\ GET #B =
          OA$MERGE__KEY:5:16\\DECIMAL I \\<->
      .FX COMPUTE #DIFF = #A - #B
         GET OA$MERGE__LINE=OA$MERGE__KEY "    " #DIFF>
```

This directive belongs on a single line but is broken here for clarity.

When you call MERGE, it processes DOTREES.BLP as follows:

1. It encounters the <&OA and treats what follows as one or more function calls, up to the first >.
2. In this case the .IF script command tests the first seven characters of each OA$MERGE__KEY, and if that string equals SPECIES, .FX calls the GET function to save the name of the species in special symbol #SPECIES and then resets special symbol OA$PAGE__LENGTH.

   In this case, resetting OA$PAGE__LENGTH to 1 causes MERGE to process a header-file (if any) as the next step.

3 If the first seven characters of OA$MERGE_KEY do not equal SPECIES, then OA$MERGE_KEY is a string consisting of a state name and two numbers. The COMPUTE function (called through the ELSE portion of the .IF statement) extracts those two numbers, subtracts the smaller from the larger, and saves the difference in local symbol #DIFF. Then, GET OA$MERGE_LINE generates the only output lines of the template.

Note the use of the double backslash in the preceding .IF statement to separate .FX function calls. If these were single backslashes, ALL-IN-1 would misinterpret the entire directive.

SPECIES.LIS and DOTREES.BLP are processed in conjunction with header-file HEADER.BLP:

```
<&NO_PAGE_1><&OA GET OA$PAGE_LENGTH="47">
<&OA .IF OA$PAGE_NUMBER NES "1" THEN .FX GET OA$MERGE_LINE="">
<&OA .IF OA$PAGE_NUMBER NES "1" THEN .FX GET OA$MERGE_LINE="">
<&OA .IF OA$PAGE_NUMBER NES "1" THEN .FX GET OA$MERGE_LINE="">
<&OA .IF OA$PAGE_NUMBER NES "1" THEN .FX <->
        GET OA$MERGE_LINE="<&TAB 28><OA$PAGE_NUMBER>">

<&PAGE>

<&TAB 10>Ranges of Selected Trees in Representative Western States

<&TAB 28>< SPECIES>
```

Note: all figures are average elevations above sea level, in feet

State Beginning of range End Avg. Extent of Range

This file numbers each page in the output file and precedes each page with the supplied header information. OA$PAGE_LENGTH is set to its usual value, 47, when MERGE processes HEADER.BLP.

In summary, the function call:

<MERGE DOTREES.BLP,OUT.LIS,SPECIES.LIS,HEADER.BLP
is processed as follows:

1 OA$PAGE_NUMBER is 0 by default and OA$PAGE_LENGTH is 57 by default before any MERGE processing begins.

2 Since there is a header-file present, MERGE looks in it first to see whether to output header information at the top of the first page. HEADER.BLP contains the <&NO_PAGE_1> directive, so the page header is not output on the first page.

3 The list and template files are opened. OA$PAGE_NUMBER is set to 1.

4 The first list-file record passes the template .IF test; that is, it begins with the word SPECIES. MERGE increments its line count by 1, so it equals 1; but the template resets OA$PAGE_LENGTH to 1, so MERGE processes the header-file after it makes this pass through the template.

5 The first directive in the header-file resets the page length to 47 lines. Thus, after MERGE makes this pass through the header-file, the page length is 57 until the next list-file line that begins with the word SPECIES.

MERGE resets its internal line counter to 0 when it encounters a <&PAGE> (or {FF}) in the header-file, and increments the counter by 1 with each line output. All lines in the header-file are included in the line count. In this example, OA$PAGE_LENGTH = 47, and there are 5 lines of text or white space before the <&PAGE>, and there are 11 lines of text or white space after the <&PAGE>.

Lines in the header-file which precede a {FF} or <&PAGE> serve as footer lines.

6 The cycle continues until MERGE processes all list-file lines. The resultant output file, OUT.LIS, is a multipage report in which each species of tree begins its own page. If the list file contained one species of tree with more data than could fit on a single page, MERGE automatically creates a page break and processes the header-file.

You can expand the capabilities of MERGE by calling MERGE_LINE from the Scripts or Named Data that you have in turn called from within a template.

10.7 Formatting and Outputting Text

MERGE templates provide one way to format text; you can also use other ALL-IN-1 features to format and output text or to process files of any sort.

When you create a document through Document Processing or a program module through Program Development, you can specify a Format, or special handling, for that document or module. A Format is a file processing method — a text editor, a postprocessor like a programming language compiler, or a combination of these or similar tools.

An indexed collection of Formats, maintained through the System Manager subsystem, provides a variety of ways to format files both during the editing process and during the output process (printing, listing, compiling).

The names of Formats serve as keys to indexed file OA$DATA:FORMAT.DAT, and the information related to the format is stored as additional fields in each record. You can access FORMAT.DAT through entry form FORMAT. (This displays when FORMAT is the current System Manager master file; see the *System Manager's Guide* for more detail.)

Figure 10-2 illustrates form FORMAT, and the paragraphs that follow explain how ALL-IN-1 uses the information contained in these fields.

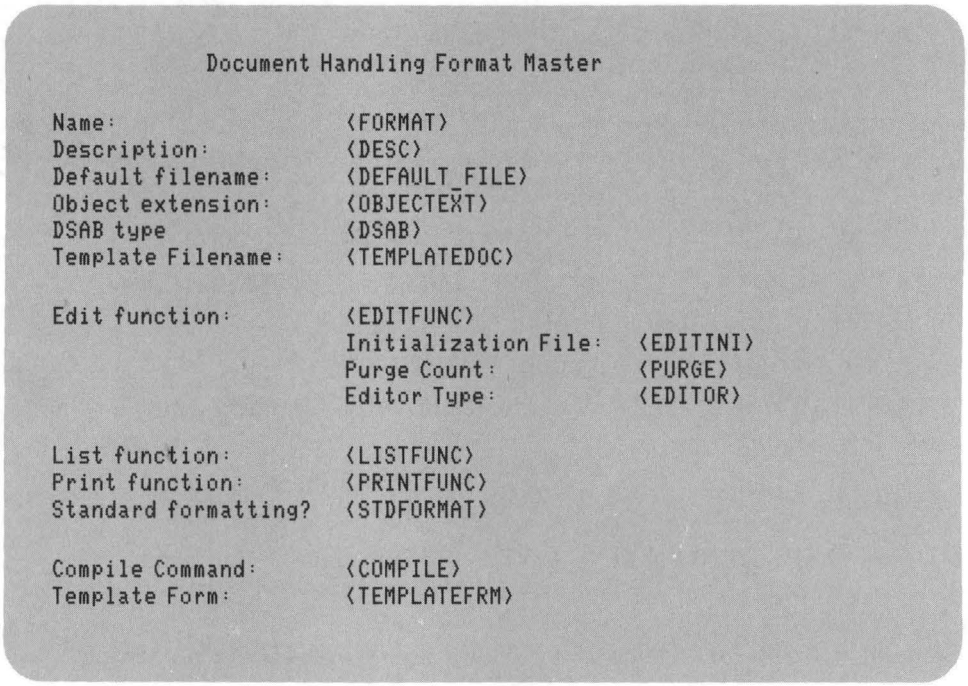


Figure 10-2 Entry Form FORMAT (Document Handling and File Processing)

FORMAT

Format Master Name; this is the key to FORMAT.DAT and the name that identifies a given format. FORMAT.DAT contains the following entries when ALL-IN-1 is installed:

- ASCII EDT — Vanilla ASCII document - no formatting - EDT style keyboard
- ASCII WPS — Vanilla ASCII document - no formatting - DECMATE style keyboard
- DECTEXT — DECtext document
- DECTYPE — DECtype document
- DICTIONARY — DECspell dictionary document

- DX — DECdx document - DECMATE/WPS-8 format
- DX8 — Extended 8 bit Multinational Character Set format
- LETTER — Business letter written to name in a directory
- MEMO — System standard memo format
- RUNOFF — DEC standard RUNOFF formatted document
- SPECIFICATION — Specification boilerplate template
- STANDARD ASCII — ASCII but with standard format processing
- WPSPLUS — WPS-PLUS Document - standard formatting
- WPSPLUS NO FMT — WPS-PLUS document - no format processing

Some of these formats, such as RUNOFF and WPSPLUS, are only meaningful if you use those products on your system since ALL-IN-1 invokes these products to edit or format a file.

DESC

Document description; the contents of this field, which describes the format, are given in the preceding list.

DEFAULT_FILE

Default File Extension; this field contains the one- to three-character VMS file-type extension that ALL-IN-1 uses to determine how to process the file.

Each recognized editor style is associated with a file-type extension. Through entries in FORMAT.DAT, ALL-IN-1 associates other file-type extensions with file processors that are not necessarily text editors.

Table 10-2, which appears later in the text, lists the recognized file-type extensions and the associated file processors.

OBJECTEXT

File processors such as RUNOFF and the programming languages produce output files that are in a different format than the source file. RUNOFF, for example, produces an ASCII-like file containing escape sequences that allow bolding and underlining; and all the language processors produce binary object modules.

Differences between output-file and input-file format are reflected by differing file-type extensions: RUNOFF input files have a default extension of .RNO, and RUNOFF output files have a .MEM extension.

Table 10-2, which appears later in the text, lists the output-file extensions associated with each file processor.

DSAB

Text data-set category; this field corresponds to the internal structure of the document.

The valid data-set categories are:

ASCII
DECTEXT
DECTYPE
DICTIONARY
DX
DX8
DXT
WPSPLUS

Most of the file-type processors that are not integrated into ALL-IN-1 (such as the language processors and RUNOFF) act upon ASCII-formatted files. Thus, the ASCII data set category is the default value for this field. However, you do not have to observe this limitation when creating new Formats. You can create a Format based on WPS-PLUS documents, for example, which automatically calls the FORMAT function to convert the file to ASCII internal format before postprocessing. This sort of approach allows users to create RUNOFF source files and program modules using the WPS-PLUS editor.

Table 10-2 lists some of the information about file processors that ALL-IN-1 stores in OA$DATA:FORMAT.DAT.

Table 10-2 File Processors

Format	DEFAULT_FILE	OBJECTEXT	DSAB
ASCII EDT	TXT	TXT	ASCII
ASCII WPS	TXT	TXT	ASCII
DECTEXT	DOC	DOC	DECTEXT
DECTYPE	DTP	DTP	DECTYPE
DX	DX	DX	DX
DX8	DX8	DX8	DX8
MEMO	TXT	TXT	ASCII
RUNOFF	RNO	MEM	ASCII
STANDARD ASCII	TXT	TXT	ASCII
WPSPLUS	WPL	WPL	WPSPLUS
WPSPLUS NO FMT	WPL	WPL	WPSPLUS

TEMPLATEDOC

Template VMS file; this field contains the VMS file name of an optinal boilerplate template that ALL-IN-1 processes with the MERGE function before the user begins editing a file. For example, if a user creates a document with the Format of MEMO, Document Processing checks whether the MEMO Format has an associated template file. Since it does, OA$LIB:MEMO.BLP, Document Processing calls MERGE to process this template and then places the user in the editor with the results.

EDITFUNC

Edit function to call; this field contains an ALL-IN-1 function to dispatch during the editing process. Usually, this is the EDIT function, but you can conceivably call other functions.

The following three fields contain information that is usally passed as parameters to the specified edit function. Thus, the contents of fields EDTINI, PURGE, and EDITOR further control the editing process.

EDITINI

Edit initialization file to invoke; the contents of this field are passed as a parameter to the specified edit function and are thus only meaningful when that edit function recognizes an initialization file parameter.

PURGE

Purge count; the contents of this field identify how many versions of a file ALL-IN-1 keeps during Formatting. The contents of this field are passed as a parameter to the edit function.

EDITOR

Editor to invoke; the contents of this field are passed as a parameter to the edit function.

LISTFUNC

List function to call; this field contains an ALL-IN-1 function used to output the formatted file to the user's terminal.

PRINTFUNC

Print function to call; this field contains an ALL-IN-1 function used to print the formatted file.

STDFORMAT

Standard formatting flag; this flag field indicates whether to use standard formatting when preparing the file.

COMPILE

VMS command to compile; for those Formats that require a compile step, such as programming languages, this field contains the VMS command that invokes the compiler. You can also specify text formatting commands such as RUNOFF here.

TEMPLATEFRM

Contains the name of a form to call before ALL-IN-1 merges with the template VMS file. For example, for a memo, ALL-IN-1 calls the form MEMARG to get the addressee and subject of the memo. Then, ALL-IN-1 merges this information into the created document using OA$LIB:MEMO.BLP.

11

Electronic Messaging

11.1 Introduction

The ALL-IN-1 Electronic Messaging subsystem enables users of single and multinode systems to communicate with each other. Electronic Messaging is accessed by a single function, MAIL, and specific tasks are implemented by calls to Mail subfunctions.

11.2 Electronic Messaging System Concepts

Electronic Messaging is built around several core concepts, which are explained in the following sections.

11.2.1 Current Message

The current message is the message most recently acted upon by Electronic Messaging. A message remains current until it is replaced by another message. CREATE, INDEX, NEXT, GET, DELETE, and OUTBOX subfunctions replace the current message.

11.2.2 Message Header and Text

A mail message is made up of two parts:

- Header information
- Text

When you create a message, all data associated with it is stored in the message header (for example, addressees, subject, date and so on). The text of the message is stored in the message text file. When you send the message, the header information determines how the message is processed. You can add addressees to the message before sending it, simply by editing the message header.

11.2.3 Mail Interfaces

Electronic Messaging provides the user with a choice of two styles of user interface, either Electronic Messaging or Electronic Messaging (DECmail).

The first is an ALL-IN-1-style interface, with menus and options that resemble the rest of ALL-IN-1. The second is a DECmail-style interface with menus and options that resemble DECmail.

11.2.4 Mail Folders

There are four special Electronic Messaging folders, which have different names depending on the user interface. The folder names are stored in local symbols as follows:

DECmail-Style Interface	ALL-IN-1-Style Interface	
Unsent folder	Created folder	#UNSENTfolder
Sent folder	Outbox	#SENTfolder
Unread folder	Inbox	#UNREADfolder
Read folder	Read folder	#READfolder

Refer to the *User's Reference,* for more information on Electronic Messaging folders.

11.2.5 Nicknames and Distribution Lists

ALL-IN-1 provides a mechanism for creating and using nicknames and distribution lists to send mail. There are menu options to create, modify, and list nicknames and distribution lists.

Each user has one nickname list, to which he or she can add or delete names.

Each user can have any number of distribution lists. They are stored in the Distribution List folder and are identified by an at sign (@) at the beginning of the name (for example, @LISTNAME) or a colon (:) at the end of the name (for example, LISTNAME:). When you create a distribution list, you can include comments about the people in the list. Precede the comments with an exclamation point (!), for example:

Mail Address:

! John Andrews
ANDREWS
! Jim Smith
SMITH
BROWN
JONES

Refer to the *User's Reference* for more information about nicknames and distribution lists.

11.2.6 Mail Delivery Mechanism

When Electronic Messaging processes a message to be sent, it scans the message header for valid addressees. Electronic Messaging determines the delivery for each message in the following way:

1. Looks up the addressee in the sender's nickname list. If it finds a matching nickname it continues the validation on the corresponding mail address.
2. Checks if the addressee is one of the following:
   a. I or ME — If so, it sends the message to the sender.
   b. PAPER MAIL or HARD COPY — If so, it prints the message on the sender's default printer.

3 Checks if the addressee is remote; that is, in the format:

name @destination @node

OR

name at destination at node

If so, it performs no further validation before sending the message.

4 Checks if the addressee begins with an at sign (@) or ends with a colon (:). If so, Electronic Messaging assumes it is a distribution list and looks at the sender's Distribution List folder. It then sends the message to everyone on the list.

If the addressee is the special distribution list SUBSCRIBERS:, MAIL sends the message to everyone with an account on ALL-IN-1.

5 Checks if the addressee begins with an underscore (_). If so, it assumes the mail destination to be SPECIAL, and invokes an alternate message transport system.

6 Looks up the addressee in the User Profile. If the mail destination of the addressee is one of the following, it then sends the message to that destination:

a ALL-IN-1 (or ALLIN1)

b VAXMAIL

c PRINTER

d HARD-COPY

If the mail destination is NO MAIL, it displays an error message to the sender. Otherwise, it invokes an alternate message transport system.

7 Looks up the address in NETWORK.DAT

See the MAIL CREATE subfunction later in this chapter for more information on addressee validation.

11.2.7 Alternative Message Transport Systems (MTS)

An alternative Message Transport System is any set of command procedures for sending electronic mail that supplements the existing MAIL function capabilities. As covered in points 5 and 6 in the preceding list, Electronic Messaging invokes an alternative MTS if:

- The mail destination is SPECIAL, that is, the address begins with an underscore.
- The mail destination in the User Profile is unrecognized (that is, neither ALL-IN-1, VAXMAIL, NO MAIL, or PRINTER).

The alternative MTS invoked when the mail destination is SPECIAL consists of two command procedures provided with ALL-IN-1; you can customize these procedures as needed. Any other MTSs are optional, site-supplied procedures.

An alternative MTS works in two parts:

1 It collects address information.

2 It processes the Send command.

The two are separate because the collecting of information may have to interact with the user (to obtain additional information), and the Send process cannot (because, as in the case of deferred mail, the user may no longer be present).

A DCL command file implements (or controls) each part of the process. ALL-IN-1 invokes the command file(s) as explained in the following sections.

NOTE: In some cases, there may be no need to collect further addressing information, and a single (Send) command file can do all the necessary processing.

Electronic Messaging looks for the command files first in the sender's directory, then in OA$LIB, so users may have different alternative MTSs to those supplied by the system manager.

11.2.7.1 SPECIAL Mail Destination. If an addressee begins with an underscore (_), Electronic Messaging invokes a pair of command procedures: SPECIAL.CMU to gather addressing information, and SPECIAL.COM to send the message. First, Electronic Messaging tries to invoke SPECIAL.CMU. The ALL-IN-1 symbol #MAILFILE contains the address string entered by the user. The command file then does the following.

1. Collects any additional addressing information (by looking up a private directory, displaying a form to the user, or whatever is necessary).
2. Concatenates all the information into a single string, using a convention of separators that the second command file, SPECIAL.COM, understands. The first character of the string *must* be an underscore (_).
3. Loads the string in temporary symbol #EMDADDRESS.
4. May add a user friendly string into the symbol #EMDADDRESSINFO. This string makes up the address that is displayed to the user, but it does not affect the final address to which the message is delivered. This string can contain no more than 30 characters.

When the user enters the Send command, Electronic Messaging again recognizes the underscore (_) and invokes the command file SPECIAL.COM. The command file obtains the contents of the following ALL-IN-1 symbols by using GETs through the MAILBOX:

- #MAILFILE
- #MAILADDR
- #MAILSUBJ

11.2.7.2 Unrecognized Mail Destination. If the mail destination for the addressee in the User Profile is not ALL-IN-1, VAXMAIL, NO MAIL, or PRINTER, Electronic Messaging uses the contents of the mail destination

field (MAIDES) to form the name of a command file, <profil.maides>.CMU, which it tries to invoke. Like SPECIAL.CMU, the command file takes as its parameter the address string typed by the user. It then does the following:

1 Collects any additional addressing information (by looking up a private directory, displaying a form to the user, or whatever is necessary).

2 Concatenates all the information into a single string. The string must begin with the addressee typed by the user, followed by a reserved separator (such as "__"), and the additional addressing information.

3 Stores the addressees in local symbol #EMDADDRESS.

4 May add a user friendly string into the symbol #EMDADDRESSINFO. This string makes up the address that is displayed to the user, but it does not affect the final address to which the message is delivered. This string can contain no more than 30 characters.

When the user enters the Send command, Electronic Messaging looks up the part of the string preceding the "__", in the User Profile. It again finds the same unrecognized mail destination, but this time it invokes the command file <profile.maides>.COM. Electronic Messaging passes the command file the following symbols:

- #MAILUSER
- #MAILFILE
- #MAILNODE
- #MAILADDR
- #MAILSUBJ

11.2.7.3 Examples of Using an Alternative MTS.

Example 1

Suppose the user enters __TELEX as an addressee:

1 Electronic Messaging invokes SPECIAL.CMU, passing it the string __TELEX as #MAILUSER.

2 SPECIAL.CMU recognizes telex and displays a form to collect the recipient's name, telex number, and answer-back.

3 SPECIAL.CMU returns a string something like:

```
__TELEX\Dave Burnett\848353\DECRDG 848353 G\\
```

(The "\" can be any character the MTS implementor chooses.)

4 Electronic Messaging stores the string in the TO or CC field of the message header.

When the user enters the Send command:

1 The message delivery code recognizes the "__", and invokes SPECIAL.COM, passing it the whole of the string above.

2 SPECIAL.COM recognizes telex, and causes the Telex server (supplied by the MTS implementor) to run, dial the number 848353, and check answer-back DECRDG 848353 G.

Example 2

Suppose the user types BURNETT

1 The validation routine looks up BURNETT in the User Profile, and finds a mail destination of TELEX.

2 The validation routine then invokes TELEX.CMU, which looks up BURNETT in a private Telex directory file.

3 If the name is not listed there, TELEX.CMU displays a form to the user to collect the necessary information (as in example 1). If the name is listed, it extracts the number and answer-back from the directory entry.

4 TELEX.CMU returns a string something like:

```
BURNETT__848353\DECRDG 848353 G\\
```

When the user enters the Send command:

1 The local message delivery code looks up BURNETT (stopping at the "__"), and again finds a mail destination of TELEX.

2 The code then invokes TELEX.COM, passing it the entire string.

3 TELEX.COM parses the string and causes the Telex call to be placed (as in example 1).

11.3 Special Symbols

See the special symbols table in Chapter 8 for a complete list of the special symbols which are used to extract information from a mail message. These Electronic Messaging special symbols begin with OA$CURMES.

11.4 Customizing the Message Header and Show-Message Displays

Four templates and the numerical value of a logical name determine what the message header looks like when the user reads or prints a message.

The following Mail-related logical-name assignment is made during installation:

- OA$MAIL_NUM_ADDRESSES

  This logical specifies the number of addressees which the addressee names display after the message text instead of before. The rest of the header information displays before the message, as determined by this file.

Following are descriptions of the templates:

- MAILMEMO1.BLP

  Electronic Messaging uses this template when the number of addresses on the main message is fewer than the number specified by OA$MAIL_NUM_ADDRESSES.

- MAILMEMO2.BLP

  Electronic Messaging uses this template when the number of addresses on the main message is greater than the number specified by OA$MAIL_NUM_ADDRESSES.

- MAILATT1.BLP

  Electronic Messaging uses this template when the number of addresses on an attachment is fewer than the number specified by OA$MAIL_NUM_ADDRESSES.

- MAILATT2.BLP

  Electronic Messaging uses this template when the number of addresses on an attachment is greater than the number specified by OA$MAIL_NUM_ADDRESSES.

In addition, another template, SHOWMEMO.BLP, formats the screen display when you call the MAIL STATUS subfunction (or the Show Message option). An additional template, RRTEMPLAT.BLP, formats the Return Receipt sent when the sender requests it and an addressee reads the message.

All of these templates are processed by the MERGE function during message processing. You may customize these templates or create new ones.

11.5 MAIL Subfunctions

The MAIL subfunction performs a single task within the Electronic Messaging subsystem. The syntax for a mail subfunction call is:

```
MAIL subfunction [argument][/qualifier[/qualifier...]]
```

where

MAIL — is the required MAIL function call,

subfunction — is the subfunction name,

argument — is the optional subfunction argument,

/qualifier — is one or more optional subfunction qualifiers.

For example:

```
MAIL FORWARD "SMITH"/PRIORITY = FIRST_CLASS
```

The following sections discuss each mail subfunction in detail.

11.5.1 Answer

ANSWER [/qualifier[/qualifier...]]

The ANSWER subfunction creates a new message using the sender of the current message as the addressee. If the original message was sent to more than one user, you have the option of replying to everyone who received it.

The subject is changed to include "Re:". The original message (that is, the message that was current when the ANSWER subfunction was issued) is saved. You can select it again by using the MAIL ORIGINAL subfunction. The subfunction takes the following qualifiers:

/ALL

The /ALL qualifier causes the answer to be sent to all the TO: and CC: recipients of the original message.

/COMMAND[=file-spec]

When the text of the answer is edited, the /COMMAND qualifier specifies the name of the command procedure to call to edit the current message. If the /NOEDIT or /EDT qualifier is also specified, /COMMAND has no effect. The default file-spec is WPEDIT.COM.

/EDT[=editor-ini-spec]

The /EDT qualifier causes the ALL-IN-1 editor to be used. When specified, the users' default editor usually used to edit the message is not called. The single, optional argument is the name of a supplemental editor initialization file.

/FNAME=file-spec

When Electronic Messaging creates a message, it generally uses a sequential number to create a unique message text file name. The /FNAME qualifier specifies the text file name to use instead.

/NOALL

The /NOALL qualifier causes the answer to be sent to the sender of the current message only.

/NOEDIT

The /NOEDIT qualifier specifies that the answer is to be created by Electronic Messaging. However, it does not place the user in the editor to enter the text of the answer. Use the MAIL EDIT subfunction to edit the text.

/NOSEND

The /NOSEND qualifier specifies that the answer is to be created but not sent. Use the SEND subfunction to send the answer.

/PRIORITY

The /PRIORITY qualifier overrides the default priority of the message. The default priority is First_Class. (See the MAIL PRIORITY subfunction for more details.)

11.5.1.1 Examples

MAIL ANSWER

Creates an answer for the current message. The editor is called so the user can enter the text of the answer. After the text is complete, the user is prompted to send the answer to the whole distribution list, or only to the original sender, then the answer is sent immediately.

MAIL ANSWER/NOALL

Creates an answer and calls the text editor so the user can enter the text of the answer. The answer is sent immediately after the text is complete, and is sent only to the original sender.

MAIL ANSWER/NOSEND

Creates an answer, calls the text editor, but does not send the answer. The answer is left in the user's File Cabinet as a CREATED message. Use the SEND subfunction to send the answer.

11.5.1.2 Special Features and Restrictions. If neither /EDT nor /NOEDIT are specified, the user's default editor (as defined in the User Profile) is called.

11.5.2 ATTACH

ATTACH [document-ID-symbol]

The ATTACH subfunction attaches a copy of an existing document or mail message to the current message.

The single parameter is the document ID (or a symbol containing the Document ID) of the message or document to be attached. If you omit the parameter, Electronic Messaging prompts for one.

11.5.2.1 Examples

```
MAIL ATTACH "SALES                    000014"
```

Attaches a copy of document number 14 in the folder called "Sales" to the current message.

```
MAIL ATTACH $DOC
```

Attaches a copy of the document contained in the symbol $DOC to the current message.

11.5.3 AUTO_FORWARD

AUTO_FORWARD [address]

The AUTO_FORWARD subfunction sends all incoming mail for the current user to a different address by setting the auto-forward address field in the User Profile.

The single parameter specifies the address to which mail is to be sent. Omit the address to cancel the auto-forward process and reset the auto-forward address field in the User Profile.

11.5.3.1 Examples

```
MAIL AUTO_FORWARD "SMITH @A1 @BOSTON"
```

Automatically forwards all incoming mail to user SMITH whose ALL-IN-1 account is on node BOSTON.

Note that "A1" may be customized.

11.5.4 AUTO__REPLY

AUTO-REPLY [document-ID-symbol]

The AUTO__REPLY subfunction specifies a document that is to be automatically sent as a reply to any mail that the current user receives. It sets the auto-reply field in the User Profile.

The single parameter is the ID of the document (or a symbol containing the Document ID) to be sent as a reply. If you omit the parameter, Electronic Messaging prompts for one.

11.5.4.1 Examples

```
MAIL AUTO__REPLY "GENERAL                    000001"
```

Sends document number 1, in the folder named "General" as a reply to any mail received.

```
MAIL AUTO__REPLY $DOC
```

Sends the document contained in the symbol $DOC as a reply to any mail received.

11.5.4.2 Special Features and Restrictions. Electronic Messaging sends the reply document only once to each sender, so if a user sends more than one message, Electronic Messaging only replies to the first one received.

11.5.5 CANCEL

CANCEL

The CANCEL subfunction cancels a posted message that has not yet been delivered (usually a deferred message). The subfunction removes from the sender queue, the entry for the message. It changes the status of the

message from SENT to UNSENT, and moves it from the Sent folder, or Outbox, to the Unsent, or Created folder.

There is no parameter. MAIL displays a list of possible documents to cancel.

11.5.5.1 Example

MAIL CANCEL

Displays a list of messages and prompts for the one you want to cancel.

11.5.5.2 Special Features and Restrictions. Messages that you cancel are normally deferred messages (see MAIL DEFER). However, occasionally there may be other messages in the queue that have not yet been delivered. You can cancel these too.

11.5.6 CANCEL_AUTO_REPLY

CANCEL_AUTO_REPLY

The CANCEL_AUTO_REPLY subfunction cancels the auto-reply process and clears the auto-reply field, MAIL_REPLY, in the User Profile.

11.5.6.1 Example

MAIL CANCEL_AUTO_REPLY

Resets the auto-reply field in the User Profile, and stops a reply from being automatically sent to any mail you receive.

11.5.7 CC

CC [/qualifier] addressee-symbol

The CC subfunction adds a carbon-copy addressee to the message being created.

The parameter specifies the addressee to include as a CC: recipient in the message. It may be an ALL-IN-1 symbol containing the addressee.

The subfunction takes the following qualifier:

/NOPROMPT

The /NOPROMPT qualifier specifies that even if CC finds the addressee is invalid, the addressee is still added to the message being created. In this case, the sender is not informed of the error.

11.5.7.1 Examples

MAIL CC "SMITH"

Checks that addressee SMITH is valid, and if it is, adds it to the current message. If SMITH is invalid, the sender is informed.

MAIL CC/NOPROMPT "SMITH"

Adds the username SMITH to the current message, even if SMITH is an invalid addressee.

MAIL CC $NEXT_NAME

Stores the addressee contained in the symbol "$NEXT_NAME" in the message.

11.5.7.2 Special Features and Restrictions. The CC subfunction validates the addressee before adding it to the current message. If it finds an error, it displays a message on the user's screen unless the /NOPROMPT qualifier is specified. If the validation invokes an Alternate Message Transport System command file, that command file may display forms or prompts to the user.

11.5.8 CLOSE_MESSAGE

CLOSE_MESSAGE

The CLOSE_MESSAGE subfunction closes the current mail message that was left open by the MAIL CREATE/OPEN subfunction. The message is then available for processing.

11.5.8.1 Examples

See the MAIL CREATE/OPEN example under the CREATE subfunction.

11.5.8.2 Special Features and Restrictions. Ensure that you close the message before trying to edit or send it.

11.5.9 CREATE

CREATE [/qualifier[/qualifier...]]

The CREATE subfunction creates a new message. The basic methods of creating a mail message are:

- Use MAIL CREATE to prompt for all header components (TO, CC, and Subject) and then edit the message text to include the text of the message. This is considered the normal method and is the easiest to call.

  When you call MAIL CREATE in this way, form EMHEAD displays by default to prompt for addressees and other recipients. Electronic Messaging implements special validation and scrolling DSRs to gather addressee information. These DSRs are covered at the end of this chapter.

- Create the message without using Electronic Messaging (Electronic Messaging has no knowledge of the message until it is sent). You can use this method in an application that creates mail messages only; the messages can be delivered by ALL-IN-1.

- Build a mail message one component at a time using the MAIL CREATE/NOSEND/OPEN qualifiers. This allows command procedures and other programs running in the subprocess to create a message without having to know the format of the message file.

The subfunction takes the following qualifiers:

/COMMAND[= file-spec]

When the text of a message is being created, the /COMMAND qualifier specifies the name of the command procedure to call to edit the current message. If the /NOEDIT or /EDT qualifiers are also specified, /COMMAND has no effect. The default file-spec is WPEDIT.COM.

/EDT[= edit-ini-spec]

The /EDT qualifier causes the ALL-IN-1 editor to be used. When specified, the users's default editor normally used to edit the message is not called. The single, optional argument, edit-ini-spec, is the name of a supplemental editor initialization file to use. Note that you cannot change the header information by editing the message text file. You must create and edit the message header separately.

/FNAME = file-spec

When Electronic Messaging creates a message, it usually creates a unique message text file name. The /FNAME qualifier specifies the text file name to use instead.

/NOEDIT

The /NOEDIT qualifier specifies that the message is to be created by Electronic Messaging, but that the user is not to be placed in the editor to enter the text of the message. Use the MAIL EDIT subfunction to edit the text.

/NOSEND

The /NOSEND qualifier specifies that the message is to be created but not sent. Use the SEND subfunction to send the message.

/OPEN

The /OPEN qualifier causes a special mode of CREATE to be entered. Usually, the CREATE subfunction creates the message header, then the message text, and closes the message text file so that it is ready for editing or sending. The /OPEN qualifier sets up the message and opens the message text file. It does not prompt for header information. It then returns while leaving the message file open. Use the TO, CC and SUBJECT subfunctions to supply the header information. Use the

CLOSE subfunction to close the message file. This method is an efficient way of building a mail message under program or DCL command-procedure control. Note that if /EDT is also specified, the ALL-IN-1 editor is called to edit the message text, after which the file is left open.

/TEMPLATE = file-spec

The /TEMPLATE qualifier supplies a layout for a mail message. The file name specified is the name of a file containing text and directives in the same format as that used in the MERGE function.

When /TEMPLATE is specified, MAIL CREATE calls the MERGE function before the user sees the message. Thus, any information contained in the template is always present when the message is edited interactively. The results of this qualifier are always placed in the message file regardless of the presence of other qualifiers.

Note that you cannot change the header information which ALL-IN-1 uses to control delivery of the message, by editing the message text file. You must create and edit the message header separately.

/PRIORITY

The /PRIORITY qualifier overrides the default priority of the message. The default priority is First_Class. (See the MAIL PRIORITY subfunction for more details.)

11.5.9.1 Examples

```
MAIL CREATE
```

Creates a new mail message. The system prompts for at least one recipient with TOs, for optional carbon-copy addressees with CCs, for the subject, and then calls the user's default editor to add the text to the message.

```
MAIL CREATE/TEMPLATE = OA$LIB:MAILBP.TXT
```

Creates a normal mail message, but instead of the standard message format, the directives in the file OA$LIB:MAILBP.TXT are used to create the heading of the message. Note that TOs and CCs are added in the standard format.

MAIL CREATE/OPEN

Creates a new mail message but does not prompt. Does not enter the editor. The following command procedure segment illustrates the use of this option:

```
$ WRITE OAMAILBOX "OA MAIL PUSH_CURMES"
$ @DCLMAILBOX:
$ WRITE OAMAILBOX "OA MAIL CREATE/OPEN/NOSEND"
$ @DCLMAILBOX:
$ WRITE OAMAILBOX "OA MAIL ""TO SMITH"""
$ @DCLMAILBOX:
$ WRITE OAMAILBOX "OA MAIL SUBJECT ""Account Balance"""
$ @DCLMAILBOX:
$ WRITE OAMAILBOX "OA MAIL TEXT ""You have $23.00."""
$ @DCLMAILBOX:
$ WRITE OAMAILBOX "OA MAIL CLOSE_MESSAGE"
$ @DCLMAILBOX:
$ WRITE OAMAILBOX "OA MAIL SEND"
$ @DCLMAILBOX:
$ WRITE OAMAILBOX "OA MAIL POP_CURMES"
$ @DCLMAILBOX:
```

11.5.10 DEFAULT_MESSAGE

DEFAULT_MESSAGE

The DEFAULT_MESSAGE subfunction searches for a message to display to the user as the default current message. The subfunction searches the mail folders in the following order:

1. Inbox, or Unread - if there is any unread mail, the oldest message becomes current.
2. Created, or Unsent - the most recently created message becomes current.
3. Read - the most recently read message becomes current.
4. Outbox - the most recently sent message becomes current.

If all four folders are empty, the previous current message (if there was one) remains current.

11.5.10.1 Example

```
MAIL DEFAULT_MESSAGE
```

Searches for a message to display to the user as the current message.

11.5.11 DEFER

```
DEFER [datetime] [/qualifier]
```

The DEFER subfunction enables the user to specify a date and time when an UNSENT message is to be sent.

The single parameter specifies the date and time when the message is to be sent (in VMS standard date format). The parameter takes the following qualifier:

/NOSEND

The /NOSEND qualifier specifies that a deferred date and time should be added to the message, but that the message should not be sent. The message is left in the Unsent folder, and can still be edited. Use the SEND subfunction to send the deferred message.

11.5.11.1 Special Features and Restrictions. When a message is deferred, its status changes from UNSENT to SENT, so the user cannot edit it. However, if the /NOSEND qualifier is used, the message remains in the Unsent folder, and can still be edited.

11.5.12 DELETE

```
DELETE [/qualifier[/qualifier...]]
```

The DELETE subfunction deletes the current message, which includes both the document header and any message text. It takes the following qualifiers:

/CLEAR

The /CLEAR qualifier indicates that the status information about the message be cleared (such as ALL-IN-1 special symbols).

Note that the status information is always cleared if the /EXPUNGE qualifier is used.

/SLIST = file-spec

The /SLIST qualifier causes the DELETE to be performed on all of the message IDs (document names) listed in the file pointed to by file-spec. Normally, this file is created by a selection list. If Electronic Messaging finds an error while processing the selection list, it stops at that point and displays a message on the user's screen.

/EXPUNGE

Usually, mail is marked for deletion and moved to the Wastebasket folder. The /EXPUNGE qualifier deletes the mail completely from the user's document data base.

11.5.12.1 Example

DELETE/CLEAR

Deletes the current message and clears the memory of the message information.

11.5.12.2 Special Features and Restrictions. If you use the /EXPUNGE qualifier, there is no way to recover the deleted message.

11.5.13 DELIVERY_RECEIPT

DELIVERY_RECEIPT

The DELIVERY_RECEIPT subfunction adds a delivery receipt request to the message. When the message is delivered to the addressee, a receipt is returned to the sender in the form of a mail message. No further action is required by the addressee. Some delivery mechanisms may not support this type of request.

DELIVERY_RECEIPT differs from the READ_RECEIPT subfunction in that the DELIVERY_RECEIPT notification is sent upon delivery, while the READ_RECEIPT notification is sent when the message is read.

11.5.13.1 Example

MAIL DELIVERY_RECEIPT

Marks the message for a delivery receipt. As soon as the message is delivered to the addressee, a mail message is returned to the sender confirming the delivery.

11.5.14 DETACH

DETACH

The DETACH subfunction reverses the MAIL ATTACH process; that is, it undoes the effect of attaching documents to the current UNSENT message. The original documents that were attached are unaffected.

11.5.14.1 Example

MAIL DETACH

Displays a list of each attachment to the current message, allowing the user to select one to detach.

11.5.14.2 Special Features and Restrictions. To detach more than one attachment, specify MAIL DETACH again.

11.5.15 DIST_COPY

DIST_COPY Listname-symbol, Listname-symbol

The DIST_COPY subfunction copies a distribution list to another file.

The first parameter is the name of an existing distribution list you want to copy. It may be an ALL-IN-1 symbol containing a list name.

The second parameter specifies the name of the distribution list that you want to copy the existing list to. It may be an ALL-IN-1 symbol containing the list name. The file must not already exist.

11.5.15.1 Examples

MAIL DIST_COPY "FOO:", "MYLIST:"

Copies distribution list "FOO:" to a list named "Mylist:".

MAIL DIST_COPY $LIST1, $LIST2

Copies the distribution list whose name is contained in the symbol $LIST1 to the list contained in the symbol $LIST2.

11.5.16 DIST_CREATE

DIST_CREATE listname-symbol

The DIST_CREATE subfunction creates a distribution list.

The single parameter specifies the name of the distribution list you want to create. It may be an ALL-IN-1 symbol containing the list name. The list must not exist already.

11.5.16.1 Examples

MAIL DIST_CREATE "SECRETARIES:"

Creates a distribution list called Secretaries:, and displays a form for the user to enter the contents of the list.

MAIL DIST_CREATE $LIST1

Creates a distribution list whose name is contained in the symbol $LIST1.

11.5.17 DIST_DELETE

DIST_DELETE Listname-symbol

The DIST_DELETE subfunction deletes a distribution list.

The single parameter specifies the list name of the distribution list you want to delete. It may be an ALL-IN-1 symbol containing the list name.

11.5.17.1 Examples

MAIL DIST_DELETE "FOO:"

Deletes distribution "FOO:"

MAIL DIST_DELETE $LIST1

Deletes the distribution list whose name is contained in the symbol $LIST1.

11.5.18 DIST_EDIT

DIST_EDIT Listname-symbol

The DIST_EDIT subfunction edits a distribution list.

The single parameter specifies the list name of an existing distribution list that you want to edit. It may be an ALL-IN-1 symbol containing the list name.

11.5.18.1 Examples

MAIL DIST_EDIT "FOO:"

Displays a form for the user to edit distribution list "FOO:"

MAIL DIST_EDIT $LIST1

Displays a form for the user to edit the distribution list whose name is contained in the symbol $LIST1.

11.5.19 DIST_LIST

DIST_LIST Listname-symbol

The DIST_LIST subfunction lists the contents of a distribution list. The user is prompted for the name of the list, and ALL-IN-1 displays the contents of the list on the user's terminal.

The single parameter specifies the list name of the distribution list whose contents you want to display. It may be an ALL-IN-1 symbol containing the name of the list.

11.5.19.1 Examples

MAIL DIST_LIST "FOO:"

Displays the contents of distribution list "FOO:"

MAIL DIST_LIST $LIST1

Displays the contents of the distribution list whose name is contained in the symbol $LIST1.

11.5.20 EDIT

EDIT [/qualifier[/qualifier...]]

The EDIT subfunction edits the current message. The user can edit the message header information and/or the text of the message. The subfunction takes the following qualifiers:

/BODY

The /BODY qualifier edits the text of the message only.

/COMMAND[= file-spec]

When the text of the answer is edited, the /COMMAND qualifier specifies the name of the command procedure to call to edit the current message. The default file-spec is WPEDIT.COM.

/EDT[= edit-ini-spec]

The /EDT qualifier causes the ALL-IN-1 editor to be used to edit the text. When specified, the user's default editor which is normally used to edit the message is not called. The single, optional argument, edit-ini-spec, is the name of a supplemental editor initialization file to use.

/HEADER

The /HEADER qualifier edits the message header information only. If you specify /HEADER, the /COMMAND and /EDT qualifiers have no effect because Electronic Messaging does not use a normal text editor to edit the header.

11.5.20.1 Examples

MAIL EDIT

Edits the current mail message using the user's default editor, as defined in the User Profile. Edits the header information, then the text.

MAIL EDIT/EDT

Edits the current mail message using the EDT editor.

MAIL EDIT/BODY

Edits only the text of the message, using the default editor.

11.5.21 EXCHANGE_CURMES

EXCHANGE_CURMES

The EXCHANGE_CURMES subfunction exchanges the current message with the message at the top of the message stack (created by the PUSH_CURMES subfunction).

11.5.21.1 Example

MAIL EXCHANGE_CURMES

Exchanges the current message with the message at the top of the message stack, so that the old current message becomes the new top of the stack, and the message originally at the top of the stack becomes the new current message.

11.5.21.2 Special Features and Restrictions. The subfunction EXCHANGE_CURMES, like PUSH_CURMES and POP_CURMES, does not affect the File-Cabinet POP and PUSH stack.

11.5.22 EXPAND_DIST_LIST

EXPAND_DIST_LIST

The EXPAND_DIST_LIST subfunction expands any distribution lists found in the message header of the current message. It replaces the name of the list with each individual name. Distribution lists must be expanded before the mail is posted.

11.5.22.1 Example

```
MAIL EXPAND_DIST_LIST
```

Replaces any distribution list names in the message header with the names of each individual person in the list.

11.5.23 FILE_ATTACHMENT

```
FILE_ATTACHMENT [/qualifier]
```

The FILE_ATTACHMENT subfunction stores a copy of any documents attached to a read message in the user's File Cabinet. This subfunction checks if there is more than one attached document. If so, it displays a list of each attachment, allowing the user to select which is to be filed. The subfunction checks if the attached document is already in the user's File Cabinet. If not, it prompts the user for a document name and folder name, makes a physical copy of the text, and stores it in the user's File Cabinet.

The subfunction takes the following qualifier:

/FORM = form

The /FORM qualifier overrides the name of the form used to prompt the user for document and folder names. The form must be an entry form which has the same record definition as the user's File Cabinet form. The default is /FORM=DOCDBI.

11.5.23.1 Examples

```
MAIL FILE_ATTACHMENT
```

Displays a list of each attachment to the current message, allowing the user to select which is to be filed. Electronic Messaging then prompts for a document name and folder name to be used to store a copy of the attachment in the File Cabinet.

11.5.23.2 Special Features and Restrictions. An attached document can only be filed from a read message. If the message has not been read, the user is asked to read it.

Electronic Messaging files copies of the attachments, so the current message still remains with all its attachments.

To file more than one attachment, you must specify FILE_ATTACHMENT again.

11.5.24 FILE_BODY

FILE_BODY [document-name-symbol, folder-name-symbol] [/qualifier]

The FILE_BODY subfunction allows the user to store the text of a message in the File Cabinet. The subfunction makes a physical copy of the text, so that the user can edit the contents if necessary.

The optional document name parameter is the name under which the document is to be filed (or a symbol containing the name). If the document name parameter is omitted, Electronic Messaging prompts for one.

The optional folder name parameter is the folder in which the document is to be filed (or a symbol containing the name). If the folder name parameter is omitted, Electronic Messaging prompts for one.

The parameters take the following qualifier:

/FORM = form

The /FORM qualifier overrides the name of the form used to prompt the user for document and folder names. The form must be an entry form which has the same record definition as the user's File Cabinet. The default is /FORM=EMFBOD.

11.5.24.1 Examples

MAIL FILE_BODY

Prompts the user for a document name and folder name to be used to store the text of the message in the File Cabinet.

MAIL FILE_BODY "JANUARY", "ACCOUNTS INFORMATION"

Stores the text of the message in a document named January and a folder named Accounts Information.

MAIL FILE_BODY $DOC, $FOLD

Stores the text of the message in a document whose name is contained in the symbol $DOC and in a folder whose name is contained in the symbol $FOLD.

11.5.25 FILE_MESSAGE

FILE_MESSAGE [docname-symbol, folder-symbol] [/qualifier]

The FILE_MESSAGE subfunction creates another pointer to a READ, SENT, or CREATED message, under the document name and folder specified by the user.

This document name parameter specifies the name under which the message is to be filed. It may be an ALL-IN-1 symbol containing the name. If the document name parameter is omitted, Electronic Messaging prompts for one.

The folder name parameter specifies the folder in which the message is to be filed. It may be an ALL-IN-1 symbol containing the name. If the folder name parameter is omitted, Electronic Messaging prompts for one.

The parameters take the following qualifier:

/FORM = form

The /FORM qualifier overrides the name of the form used to name the current message. The form must be an entry form which has the same record definition as the user's File Cabinet form.

11.5.25.1 Examples

MAIL FILE_MESSAGE

Prompts for filing information, and stores a new document header in the user's File Cabinet.

MAIL FILE_MESSAGE "JANUARY", "ACCOUNTS INFORMATION"

Stores a new document header with the name January in a folder named Accounts Information.

MAIL FILE_MESSAGE $DOC, $FOLD

Stores a new document header with the name that is contained in the symbol $DOC and a folder name that is contained in the symbol $FOLD.

11.5.25.2 Special Features and Restrictions. The copy produced by FILE_MESSAGE cannot be edited, because it is not a physical copy. To produce an editable copy, you must use FILE_BODY instead.

11.5.26 FORWARD

FORWARD [addressee] [/qualifier[/qualifier...]]

The FORWARD subfunction creates a cover note, attaches the current message to it, and sends a copy to another user. The subfunction may be used to forward a SENT message to additional users.

The single optional parameter is the name of a user to forward the message to. If the addressee parameter is omitted, Electronic Messaging prompts for at least one forwarding addressee. The parameter takes the following optional qualifiers:

/COMMAND[= file-spec]

When the text of the cover note is being created, the /COMMAND qualifier specifies the name of the command procedure to call to edit the cover note. If the /NOEDIT or /EDT qualifiers are also specified, /COMMAND has no effect. The default file-spec is WPEDIT.COM.

/EDT[= edit-ini-spec]

The /EDT qualifier causes the ALL-IN-1 editor to be used. When specified, the user's default editor normally used to edit the message is not called. The single, optional argument, edit-ini-spec, is the name of a supplemental editor initialization file to use.

/NOEDIT

The /NOEDIT qualifier specifies that the cover note is to be created by Electronic Messaging, but that the user will not be placed in the editor to enter the text of the message. The text may be edited at a later time, for example, by using the MAIL EDIT subfunction.

/NOSEND

The /NOSEND qualifier specifies that the cover note should be created and the current message attached to it, but not sent. Use the SEND subfunction to send the message to be forwarded.

/OPEN

The /OPEN qualifier causes a special mode of FORWARD to be entered. Typically, the FORWARD subfunction creates the cover note and closes the message text file so that it is ready for editing or sending. The /OPEN qualifier sets up the cover note and opens the message text file. It then returns while leaving the message file open. The message file can be closed using the CLOSE subfunction. This method is an efficient way of building a mail message under program or DCL command-procedure control.

/PRIORITY

The /PRIORITY qualifier overrides the default priority of the message. The default priority is First_Class. (See the MAIL PRIORITY subfunction for more details.)

11.5.26.1 Examples

```
MAIL FORWARD
```

Prompts for one or more user names and forwards the current message to those users.

```
MAIL FORWARD "SMITH"
```

Forwards the current message to user SMITH.

```
MAIL FORWARD "SMITH"/PRIORITY = Express
```

Places the current message at the top of the sender queue, to be forwarded to user SMITH.

11.5.27 FORWARDABLE

FORWARDABLE

The FORWARDABLE subfunction sets the forwardable attribute in the message header of a message. The recipient(s) of that message can then forward it to another user.

11.5.27.1 Example

MAIL FORWARDABLE

Makes forwardable a message that was previously marked as nonforwardable.

11.5.27.2 Special Features and Restrictions. A user can always forward a message that he has created, even if it is marked nonforwardable.

11.5.28 FUNCTION

FUNCTION "function"

The FUNCTION subfunction specifies a single function to be performed when the recipient first reads the mail message. A function is only performed if it is known as a valid function on the recipient's ALL-IN-1 system. Valid functions are defined in the table MAILFUN that is found in OAGBL.BLI

The parameter specifies the name of the function to be used.

11.5.28.1 Example

MAIL FUNCTION "DISPLAY Hello John and Jim"

If the function DISPLAY is in the valid functions table on the recipient's ALL-IN-1 system then Hello John and Jim displays at the end of the message, when it is read for the first time.

11.5.28.2 Special Features and Restrictions. This subfunction is useful for other applications that use Electronic Messaging as a transport mechanism, such as CALENDAR.

11.5.29 GET

GET [document-ID-symbol] [/qualifier]

The GET subfunction allows the user to retrieve an existing message or document from the File Cabinet and make it the current one.

The parameter specifies the document ID (or a symbol containing the document ID) of the document or message to be retrieved. It can be a symbol containing the message or document. If you omit the parameter, Electronic Messaging prompts for a document ID.

The subfunction takes the following qualifier:

/FORM = form

Use the /FORM qualifier to specify the form which prompts for the name of the message. If the document ID parameter is present, this qualifier is ignored. The default is /FORM=EMGARG.

11.5.29.1 Examples

MAIL GET

Prompts the user for a message name and makes that message current.

MAIL GET OA$CURMES_ORIG_MSGID

Gets the message identified as the original message.

MAIL GET "CREATED 000003"

Gets the document number 3 in the Created folder.

11.5.30 INDEX

INDEX [Folder1, Folder2, Folder3]

The INDEX subfunction displays a list of messages and documents in one or more folders. From the list, you can select a document or message to be current.

The Folder parameters specify the folder(s) whose contents you want to list. If you omit this parameter, Electronic Messaging displays a list of the contents of the current folder.

11.5.30.1 Examples

```
MAIL INDEX "CREATED", "OUTBOX"
```

Lists all the messages in the Created folder, and the Outbox. Prompts you to select a current message.

11.5.31 MAIL_DOCUMENT

```
MAIL_DOCUMENT document-ID-symbol [/qualifier[/qualifier...]]
```

MAIL_DOCUMENT subfunction identifies a document, a copy of which becomes the text of a message to be sent. It prompts for header information to be attached to the document, then allows you to edit the document.

The parameter specifies the document ID (or a symbol containing the document ID) of the document to become the text of the message. If you omit this parameter, Electronic Messaging uses the current document. The parameter takes the following qualifiers:

/COMMAND[= file-spec]

The /COMMAND qualifier specifies the name of the command procedure to call to edit the text of the document. If the /NOEDIT or /EDT qualifiers are also specified, /COMMAND has no effect. The default file-spec is WPEDIT.COM.

/EDT[= edit-ini-spec]

The /EDT qualifier causes the ALL-IN-1 editor to be used. When specified, the user's default editor normally used to edit the document is not called. The single, optional argument, edit-ini-spec, is the name of a supplemental editor initialization file to use.

/FORM = form

The /FORM qualifier overrides the name of the form used to supply the header information for the document to be mailed. The default is /FORM=EMMARG.

/NOEDIT

The /NOEDIT qualifier specifies that the document is to become the text of the message, but that the user will not be placed in the editor. Use the MAIL EDIT subfunction to edit the text.

/NOSEND

The /NOSEND qualifier specifies that the document should become the text of the message, but the message should not sent. Use the SEND subfunction to send the message.

/OPEN

Normally, the MAIL_DOCUMENT subfunction makes the document the text of a message, and closes the message text file so that it is ready for editing. The /OPEN qualifier specifies that the user is not placed in the editor. It sets up the text and opens the message text file. It then returns while leaving the message file open. The message file can be closed using the CLOSE subfunction. This method is an efficient way of building a mail message under program or DCL command-procedure control.

/PRIORITY

The /PRIORITY qualifier overrides the default priority of the message. The default priority is First_Class. (See the MAIL PRIORITY subfunction for more details.)

11.5.31.1 Examples

```
MAIL MAIL_DOCUMENT
```

Makes a copy of the current document into the text of a message. Prompts for header information to be attached.

```
MAIL MAIL_DOCUMENT "SALES                    000003"
```

Makes a copy of document number 3 in the folder called "Sales" into the text of a message. Prompts for header information to be attached, and places the user in the editor. Sends the message as soon as the user exits the editor.

MAIL MAIL_DOCUMENT $DOC

Makes a copy of the document contained in the symbol $DOC into the text of a message.

MAIL MAIL_DOCUMENT $DOC/NOSEND

Makes a copy of the document identified into the text of a message, prompts for header information to be attached, places the user in the editor, but does not send the message when the user exits the editor.

11.5.31.2 Special Features and Restrictions. Electronic Messaging makes a copy of the document to be used as the text of a message, so the user can still edit the original document without affecting the message.

11.5.32 NEXT

NEXT [/qualifier]

The NEXT subfunction selects the next sequential entry in the Inbox as the current message. If there are no Inbox messages, the current message is unchanged, and the message `No More Messages` displays. If there are more messages, but the user has reached the last record in the Inbox (this can occur when the user does a NEXT and does not read the message), the message `Bottom of Inbox Reached` displays.

The subfunction takes the following qualifier:

/SLIST = file-spec

The /SLIST qualifier specifies that instead of selecting the next message in the Inbox, the next message named in the selection list pointed to by file-spec is selected.

11.5.32.1 Examples

MAIL NEXT

Selects the next message from the Inbox.

MAIL NEXT/SLIST = WPLIST.SEL

Selects the message from the selection list WPLIST.SEL.

11.5.33 NO_DELIVERY_RECEIPT

NO_DELIVERY_RECEIPT

The NO_DELIVERY_RECEIPT subfunction deletes a delivery receipt request from the message. By default, no receipt is sent when mail is delivered.

11.5.33.1 Examples

MAIL NO_DELIVERY_RECEIPT

Deletes a delivery receipt request from a message. The sender is not informed when the message is delivered to the addressee.

11.5.34 NON_FORWARDABLE

NON_FORWARDABLE

The NON_FORWARDABLE subfunction resets the forwardable attribute in the message header of a message, preventing that message from being forwarded by the recipient(s).

11.5.34.1 Example

MAIL NON_FORWARDABLE

Prevents a message from being forwarded by the recipient(s).

11.5.34.2 Special Features and Restrictions. The NON_FORWARDABLE subfunction only prevents the recipient(s) using FORWARD. They can still forward a message by copying it first.

11.5.35 ORIGINAL

ORIGINAL

The ORIGINAL subfunction is used with the Answer and Forward options. It makes current the message that you were answering or forwarding.

11.5.35.1 Example

MAIL ORIGINAL

Makes current the message that you were answering or forwarding.

11.5.36 OUTBOX

OUTBOX

The OUTBOX subfunction displays a list of messages stored in the Outbox. From the list, you can select a message to be current.

11.5.36.1 Example

MAIL OUTBOX

Lists all messages in the Outbox. Prompts you to select a message to be current.

11.5.37 POP_CURMES

POP_CURMES

The POP_CURMES subfunction restores the last message that was saved using the PUSH_CURMES subfunction. The message that is popped from the stack is selected as the current message. If there are no more messages on the stack, the current message is left unchanged.

The POP_CURMES and the PUSH_CURMES subfunctions must always be paired. The POP_CURMES subfunction reverses the actions of the last PUSH_CURMES subfunction.

11.5.37.1 Examples

MAIL POP_CURMES

Removes the last pushed message from the message stack and makes it the current message.

11.5.37.2 Special Features and Restrictions. The POP_CURMES subfunction, like PUSH_CURMES, does not affect the document database stack.

11.5.38 PRINT

PRINT [/qualifier[/qualifier...]]

The PRINT subfunction sends the current message to a printer. The message to be printed is copied to a temporary file and then submitted to the user's default print queue specified in the User Profile or to the queue specified in the/QUEUE qualifier.

If the message being read contains a read receipt request, and the recipient indicates that one may be sent, a mail message is returned to the sender the first time the message is printed or read.

If the message contains a function request, the function is executed the first time the message is printed or read (see the MAIL FUNCTION subfunction).

The subfunction takes the following qualifiers:

/COPIES = n

The /COPIES qualifier specifies the number of copies that are to be printed. The default is /COPIES=1.

/NOATTACHMENTS

The /NOATTACHMENTS qualifier specifies that the mail is to be printed without its attachments.

/NOHEADINGS

The /NOHEADINGS qualifier specifies that only the text of the mail is to be printed.

/QUEUE = name

The /QUEUE qualifier overrides the default queue name stored in the User Profile.

/SLIST = file-spec

The /SLIST qualifier specifies a list of documents or messages that are to be printed.

11.5.38.1 Examples

MAIL PRINT

Prints the current message on the user's default printer.

MAIL PRINT/COPIES = 3

Prints three copies of the current message on the user's default printer.

MAIL PRINT/SLIST = WPLIST.SEL

Prints the documents or messages in the selection list WPLIST.SEL on the user's default printer.

11.5.39 PRIORITY

PRIORITY [priority]

The PRIORITY subfunction changes the default priority of an UNSENT message.

The parameter is one of the following keywords:

Express

Express specifies that the message is to be delivered to addressees on the same computer system immediately, and is to be passed to the Message Router for delivery to other computer systems immediately. The message is also placed at the top of the Message Router queue.

Note that the user must wait while this happens, so Express mail may appear to be slower than the other priorities.

First_Class

First_Class specifies that the message is to be delivered to addressees on the same computer system immediately, but is to be placed on a system batch queue to be passed to the Message Router for delivery to remote computer systems. The message is also given an intermediate Message Router priority.

Note that the user can continue before this is completed, so First_Class mail may appear to be quicker than Express mail.

Second_Class

Second_Class specifies that the message is to be placed on a system batch queue for delivery to all addressees. It is also placed at the end of the Message Router queue for delivery to remote computer systems.

Note that Second_Class mail appears to the user to be the quickest, but the mail has not necessarily been delivered.

11.5.39.1 Example

```
MAIL PRIORITY Express
```

Places the current message at the top of the sender queue.

11.5.39.2 Special Features and Restrictions. You can only change the priority of an UNSENT message.

You can also change the default priority of a message by adding the /PRIORITY switch to selected Mail subfunctions. For example, SEND, ANSWER and FORWARD.

11.5.40 PUSH_CURMES

```
PUSH_CURMES
```

The PUSH_CURMES subfunction stores the name of the current message on a stack. That current message is not changed, but if another message is selected as current, the message pushed can be restored using the POP_CURMES subfunction.

The message stack is limited to 10 messages.

The PUSH_CURMES and POP_CURMES subfunctions are normally used by procedures which perform a series of mail subfunctions that do not want to destroy the status of the current message.

11.5.40.1 Example

See the CREATE subfunction for examples of how PUSH_CURMES and POP_CURMES might be used.

11.5.40.2 Special Features and Restrictions. The POP_CURMES subfunction, like PUSH_CURMES and EXCHANGE_CURMES, does not affect the document database stack.

11.5.41 READ

READ [document-ID-symbol] [/qualifier[/qualifier...]]

The READ subfunction displays the contents of the current message. It can be used with both sent and received messages. When an unread message is read, it is moved to the user's Read folder. This subfunction may be repeated as many times as needed.

If the message being read contains a read receipt request, and the recipient indicates that one may be sent, a mail message is returned to the sender the first time the message is read or printed.

If the message contains a function request, the function is executed the first time the message is printed or read (see the MAIL FUNCTION subfunction).

The parameter specifies the document ID (or a symbol containing the document ID) of the message to display. If you omit the parameter, Electronic Messaging displays the current message. The parameter takes the following qualifiers:

/NEXT

The /NEXT qualifier selects the next document or message in the current folder.

/SLIST = file-spec

The /SLIST qualifier specifies a list of documents or messages that are to be read.

/UNREAD

The /UNREAD qualifier must be used in conjunction with /NEXT to select the next UNREAD message from the Inbox.

11.5.41.1 Examples

MAIL READ

Reads the current message. If it is UNREAD, MAIL READ copies it to the user's File Cabinet and removes it from the Inbox, or Unread folder, to the Read folder.

MAIL READ/NEXT

Reads the next message from the current folder.

MAIL READ/UNREAD/NEXT

Reads the next message from the Inbox. If a read receipt is requested, it is sent (providing you indicated that one is to be sent).

11.5.42 READ_RECEIPT

READ_RECEIPT

The READ_RECEIPT subfunction adds a read receipt request to the message. When the addressee reads the message for the first time, a receipt is returned to the sender if the addressee has indicated that one may be sent.

11.5.42.1 Examples

MAIL READ_RECEIPT

Updates the current message to include a read receipt request.

11.5.42.2 Special Features and Restrictions. A read receipt is sent only if the recipient indicates that one may be sent. Each user can specify in the User Profile whether a read receipt is to be sent when requested. The user can also specify that he wishes to be asked each time he receives a read receipt request.

11.5.43 SEND

SEND [Parameter]

The SEND subfunction sends the current created message. The message header is scanned for addressees, and duplicates are eliminated.

The parameter is either the document ID of the message to be sent, or a VMS file name. If you omit this parameter, Electronic Messaging sends the current message.

If the parameter is a VMS file name, the VMS message file is sent to the addressees specified within it. This feature can be used to send a message which has been prepared by another facility or application running in the subprocess. It is the responsibility of the preparer of the message file to include the proper header information (from, to, subject, and so on). See Special Features and Restrictions for an example of correct header information.

When used in this manner, Electronic Messaging does not affect the current message.

11.5.43.1 Examples

```
MAIL SEND
```

Sends the current message.

```
MAIL SEND "CREATED                    000005"
```

Sends message number 5 in the Created folder.

11.5.43.2 Special Features and Restrictions. Messages created using ALL-IN-1 Version 1 do not have separate header information and text. Both the header information and the text are part of a message buffer. The SEND subfunction sends messages in this format as well as the ALL-IN-1 Version 2.0 format.

This is an example of a message in ALL-IN-1 Version 1 format:

```
                    I N T E R O F F I C E   M E M O R A N D U M

                                        Memo:   [45640.5744479.JONES
                                        Date:   Wed 2-NOV-1983 15:57
                                        From:   Cathy Jones
                                        Dept:   SALES
                                         Tel:   39854

TO:   Jim Saunders (TRON::SAUNDERS)
CC:   Andrea Cartwright (CARTWRIGHT)
Return-Receipt requested

Document Enclosed:
Doc-Name:      AGENDA
Doc-Title:     Agenda for meeting to be held tomorrow
Doc-Type:      DOCUMENT
Doc-Author:    Cathy Jones
Doc-Folder:    General
Doc-Setup:
Subject:       Tomorrow's meeting

I'm planning to hold a meeting tomorrow at 3 o'clock. Please let me know if
this is convenient.

Cathy
```

Apart from the text of the message, Electronic Messaging recognizes only lines beginning:

- TO:
- CC:
- Return-Receipt requested
- Doc-Name:
- Subject:

These lines must be formatted as shown in the example. The Subject line must be the last line of the header.

If some of the addressees in a TO: or CC: line are shown in parentheses (as in the example), Electronic Messaging takes that part as the address and ignores the rest.

If there are no parentheses, Electronic Messaging takes the whole of the line after the colon (:) as the address.

11.5.44 SHOW

SHOW Listname-symbol

The SHOW subfunction lists the contents of a distribution list. It performs exactly the same function as the DIST_LIST subfunction. ALL-IN-1 displays the contents of the list on the user's terminal.

11.5.44.1 Examples

MAIL SHOW "FOO:"

Displays the contents of the distribution list whose list name is "FOO:".

11.5.45 STATUS

STATUS

The STATUS subfunction displays a form showing information about the status of the current message.

11.5.45.1 Example

MAIL STATUS

Displays the following kind of information about the current message:

```
Jan Jones                                                    Mon 10-Dec-1984
                          Showing ALL-IN-1 message

Subject :           Sample Mail Message
Title :             Sample Mail Message
Sender :            JONES
Senders Full Name :Jan Jones
Senders Tel No :                              State :              SENT
Current Folder :    OUTBOX                    Document Number :    000031
Type of document :  MAIL                      Modifiable:          N
Date Created :      10-Dec-1984 01:26p        Deletable:           Y
Last Modified :     10-Dec-1984 01:26p        Forwardable :        YES
Date Sent :         10-Dec-1984 01:31p        Read Receipt :       NO
Date Posted :       10-Dec-1984 01:31         Delivery Receipt :   NO
Deferred Date :
MR id :             19841210083533/29090ATLAST
Priority :          FIRST_CLASS               No of Attachments :  0
VMS filename :      OA$SHARE2:ZQCYBBSMQ.TXT   Text type :          ASCII


Addressees :

  TO  Mike Burnett                  ( BURNETT                    )
  TO  Clive Smith                   ( SMITH                      )

  CC  Alan Pearce                   ( PEARCE                     )
  CC  Bob James                     ( JAMES                      )
```

11.5.46 SUBJECT

SUBJECT text-symbol

The SUBJECT subfunction adds a subject to the current message being created by CREATE/OPEN.

If this is the first time a subject has been added, it also becomes the title of the message.

The parameter is the subject text (or a symbol representing the subject) to be added to the message.

11.5.46.1 Examples

MAIL SUBJECT Grass seed sale

Adds the subject "Grass seed sale" to the current message.

11.5.47 TEXT

TEXT text-symbol

The TEXT subfunction adds text to a message being created using MAIL CREATE/OPEN.

The parameter is the line of text (or a symbol containing that text) to be added to the message text. If you omit the parameter, Electronic Messaging adds a blank line to the message text.

11.5.47.1 Examples

MAIL TEXT

Adds a blank line to the message text.

MAIL TEXT "Hi there"

Adds the text Hi there to the text of the message.

MAIL TEXT $MSG__LINE

Adds the contents of the symbol $MSG__LINE as a single line in the message.

11.5.48 TO

TO [/qualifier] addressee-symbol

The TO subfunction adds a new addressee to the message being created.

The parameter specifies the addressee to include on a TO: line in the message. It may be an ALL-IN-1 symbol.

The subfunction takes the following qualifier:

/NOPROMPT

The /NOPROMPT qualifier specifies that even if TO finds the addressee is invalid, the addressee is still added to the message being created. In this case, the user is not informed of the error.

11.5.48.1 Examples

MAIL TO "SMITH"

Checks that the username SMITH is valid,and if it is, adds it to the current message. If SMITH is invalid, the sender is informed.

MAIL TO/NOPROMPT "SMITH"

Adds the username SMITH to the current message, even if SMITH is an invalid addressee.

MAIL TO $NEXT_NAME

Stores the addressee contained in the symbol $NEXT_NAME in the message header.

11.5.48.2 Special Features and Restrictions. The TO subfunction validates the addressee before adding it to the current message. If it finds an error, it displays a message on the user's screen unless the /NOPROMPT qualifier is specified. If the validation invokes an Alternate Message Transport System command file, that command file may display forms or prompts to the user.

11.5.49 VAXMAIL_CHECK

VAXMAIL_CHECK

The VAXMAIL_CHECK subfunction reads the user's VAXmail message file, MAIL.MAI, stores any new messages in the user's Inbox, or Unread folder, and removes those messages from the VAXmail message file. If the ALL-IN-1 user name does not match the VMS user name, the sender name

and subject of the message displays on the terminal, and the user is asked whether to have this message included in the Inbox. If not, the message is left in the VAXmail message file.

11.5.49.1 Examples

```
MAIL VAXMAIL__CHECK
```

Stores any new VAXmail messages in the user's Inbox, or Unread folder and deletes them from the VAXmail message file.

11.6 Electronic Messaging Data Set References

Electronic Messaging implements special validation and scrolling data set references to gather information about message recipients. The following entries from the Named Data of form EMHEAD illustrate the use of these DSRs:

```
5 TO
/OPTIONAL/VALID=OA$MAIL__ADDRESS/SCROLL=,,,MAIL$SCL__FUNCTION:TO

6 TO
/RECOG="OA$MAIL__ADD__ADDR"

7 TO
/SHOW=' "( " .FULNAM:30 " )" '/USE__FORM=EMSREC

.
.
.

11 PRIORITY
/VALID=OA$MAIPRIO
```

/VALID=OA$MAIL__ADDRESS checks the contents of field for special prefixes, suffixes, and entire entries that categorize a recipient.

/RECOG="OA$MAIL__ADD__ADD" provides a list of all possible addressees known to Electronic Messaging by opening and displaying the contents of the following data set:

- OA$MAIL__ADD__ADDR – All local addressees for this user

The field qualifiers on field TO also illustrate the use of /RECOGNITION in conjunction with /VALID. In this case, /RECOGNITION provides a list of of all valid known recipients; and /VALID ensures that if the user specifies an unknown recipient, that recipient is entered in a valid format.

Although not shown in the preceding example, the same DSRs are specified on the CC field.

/VALID=OA$MAILPRIO validates against an internal table to ensure that the user enters a valid priority in the qualified field. As covered under the MAIL PRIORITY subfunction, the valid priorities are:

- Express
- First_class
- Second_class

12

External Communications

12.1 Overview

VMS supports several products that enable a VAX to communicate with a wide range of other machines. The most versatile of these products is DECnet, which links two or more computers in a transparent network. A third product, DECdx, is a specialized utility for transferring files between a VAX and a WPS-8 word processor.

12.2 Multinode ALL-IN-1 Applications

Most DCL commands that accept RMS file specifications as parameters recognize node names as valid prefixes. Thus, when DECnet is installed, most tasks that can be done locally can also be done across network links. Those commands that do not work across network links can often be replaced with other commands taking different parameters, without a loss of functionality.

When ALL-IN-1 runs in a DECnet environment, many multinode features can be incorporated into new or existing applications. Cross-node DCL commands can be called in command procedures, and these command procedures can be invoked with the COMMAND function and integrated into an application.

Also, the Electronic Messaging subsystem incorporates several multinode features and provides an interface to any alternate Message Transport System (multinode or otherwise) that you wish to provide.

The *System Manager's Guide* covers multinode ALL-IN-1 in more detail.

12.3 The Communications Control Subsystem

The Communications Control subsystem enables users to connect to and use computer systems other than the VAX running ALL-IN-1. Communications Control is accessed by a single function, CXP, and specific tasks are implemented by calls to CXP subfunctions.

12.3.1 Communications Control Concepts

Choose between the following ways of using Communications Control:

- Terminal Emulation

  You connect to a computer system and then type in commands, as if your terminal were connected directly to the computer system.

- Control Documents

  You can automate anything you can do under terminal emulation using Control Documents. Control Documents contain a set of commands that specify a task. Control Document commands are described in detail in the *User's Reference.*

Communications Control uses standard ALL-IN-1 File Cabinet documents, you can:

- Create and use Control Documents.
- Record a session with another computer system in a document.
- Send a document to another computer system.

12.3.2 CXP Symbols

CXP uses the symbols in the following tables.

Symbols in the User's Permanent Symbol Table

Symbol	Description
$CXP_CONTROL_DOCUMENT	Holds the name of the user's currently selected Control Document.
$CXP_RECORD_DOCUMENT	Holds the name of the document currently selected for recording.
$CXP_SEND_DOCUMENT	Holds the name of the document currently selected for sending.
$CXP_HOST	Holds the default value for the entry form CXP$SEL_HOST.

NOTE: These symbols are held from session to session in each user's permanent symbol table.

Symbols in the Local Symbol Table

Symbol	Description
#CXP_WAIT_MESSAGE	Holds the text of the Stop Waiting for text and Stop Pausing options from the Interrupt menu (CXP$INT1) if a string is being waited for or a PAUSE command is active.
#CXP_HOST_MESSAGE	Holds the text of the Disconnect from Computer System option which appears on the Communication Control Interrupt menu (CXP$INT1) whenever applicable.
#CXP_SEND_MESSAGE	Holds the text of the Stop Sending Document option which appears on the Communication Control Interrupt menu (CXP$INT1) whenever applicable.
#CXP_HOST_NAME	Holds the name of the computer system currently connected to. If this symbol is empty, no computer system is connected.

(continued)

Symbols in the Local Symbol Table (Cont.)

Symbol	Description
#CXP_CONTROL_MESSAGE	Holds the text of the Stop Processing Control Document option which appears on the Communication Control Interrupt menu (CXP$INT1) whenever applicable.
#CXP_VERIFY	Holds the setting of Verify Mode; either yes or no. The user can set Verify Mode to yes or no using either the Control Document Command SET or the Set Terminal menu. (CXP$OPTIONS).
#CXP_DISPLAY	Holds the setting of Display Mode; either yes or no. The user can set Display Mode to yes or no using either the Control Document Command SET or the Set Terminal menu.
#CXP_KEYBOARD	Holds the setting of Keyboard Mode; either yes or no. The user can set Keyboard Mode to yes or no using either the Control Document Command SET or the Set Terminal menu.
#CXP_INT1_PROMPT	Holds the prompt that appears if the form CXP$INT1 is used in Hard Copy Mode.
#CXP_RECORD_PROMPT	Holds the prompt that appears if the form CXP$RECORD is used in Hard Copy Mode.
#CXP_RA_MESSAGE	Holds the text of the Record/Append option that appears on the Record Session menu (form CXP$RECORD) when the user is not currently recording the session.
#CXP_RO_MESSAGE	Holds the text of the Record/Overwrite option that appears on the Record Session menu (form CXP$RECORD) when the user is not currently recording the session.
#CXP_SR_MESSAGE	Holds the text of the Stop Recording option that appears on the Record Session menu (form CXP$RECORD) when the user is currently recording the session.
#CXP_WAKUP_MESSAGE	Holds the text of the Stop Processing Wake-up Commands option which appears on the Communication Control Interrupt menu (CXP$INT1) whenever applicable.

NOTE: *These symbols are not held from session to session.*

12.3.3 CXP Subfunctions

A CXP subfunction performs a single task within the Communications subsystem. You can invoke these subfunctions either through the normal ALL-IN-1 function-processing channels or from within Control Documents. For example, the Control Document command:

```
DISPLAY "text"
```

displays whatever is specified (text). Use the same syntax to invoke this command as a CXP subfunction; for example:

```
<CXP DISPLAY "Hello"
```

12.3.3.1 Subfunction Descriptions.

```
AT "document_point" [SEND "send_string"] [WAIT command]
```

AT defines characters to send to the remote computer system at specified points in the current (or next) document being sent to the computer system. The document_point is a keyword identifying a location in the document, and the send_string is either another keyword or a string of characters to insert at the specified point. The optional SEND and WAIT parameters are both CXP subfunctions that you call separately.

Each document_point is associated with a default send_string that is used if you do not specify the AT document_point SEND send_string command. The document_point keywords and associated send_strings are:

Document_Point	Explanation	Default send_string
SOD	Start of document	""
EOD	End of document	""
SOL	Start of line	""
EOL	End of line	"<CR>" (carriage return)
EOP	End of page	"<FF>" (form feed)
PARA	End of paragraph	"<CR>" (carriage return)

```
CONNECT "host name" [/[NO]INTERACTIVE]
```

This subfunction connects you to the specified computer system. The host_name must be present in the computer system's master file,

OA$DATA:CXPHOSTS.DAT. The optional /NOINTERACTIVE qualifier prevents you from entering Terminal Emulation Mode and allows you to call other CXP subfunctions after CONNECT.

CONNECT/DEVICE "device_name" [/qualifier...][/[NO]INTERACTIVE]

This command connects to the computer system using the information provided with the qualifiers. See the *System Manager's Guide* for details of how each qualifier should be set. Each qualifier matches a field on either the Define Computer systems or Define Communications Lines form. This command is useful for connecting to computer systems that have not been entered on the Define Computer Systems form.

Device name is the device name of the terminal line and N is a numeric character. Specify the /NOINTERACTIVE qualifier if you want to connect to the specified computer system without entering Terminal Emulation Mode. The default is /INTERACTIVE.

The optional qualifiers are:

/LINE_SPEED = {110|300|600|1200|1800|2000|2400|3600|4800|7200|9600}
/CONNECTION_TYPE = {"HARDWIRED"|"MANUAL DIAL"|"AUTODIAL"}
/MODEM_TYPE = {DF100P|DF100T|DF02AC|DF03AC|VADIC811|MPS1222AD
|CASE440}
/[NO]ECHO
/[NO]BUFFER_CONTROL
/INT_CHARACTER = {A|B|D|E|F|G|H|I|J|K|L|N|O|P|R|U
|V|W|X|Z}
/BR_TIME = NNN
/BR_KEY = {A|B|D|E|F|G|H|I|J|K|L|N|O|P|R|U|V|W|X|Z}
/PHONE_NUMBER = NNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

DISCONNECT

This subfunction ends the session on the computer system.

DISPLAY "text"

This subfunction displays the contents of text on the user's terminal.

PAUSE n

This subfunction causes a pause of n-hundredths of a second.

PROCESS [/FOLDER = "folder"] [/NUMBER = N] ["name"] [/TEST]
[/[NO]INTERACTIVE]

This subfunction process the Control Document identified by the File-Cabinet title name. All Control Documents are stored in the ALL-IN-1 File Cabinet, and the /FOLDER and /NUMBER qualifiers further identify the document. The /TEST qualifier causes a syntax check to be made on the Control Document.

If you do not want to remain in Terminal Emulation Mode after processing the Control Document, use the /NOINTERACTIVE qualifier. The default is /INTERACTIVE.

PROCESS/VMS/[[NO]INTERACTIVE] vms_file_name

This subfunction works like the PROCESS subfunction, but it uses a VMS file instead of a file in the ALL-IN-1 File Cabinet.

RECORD[/FOLDER = "folder"][/NUMBER = N][{/APPEND|/OVERWRITE}] ["name"]

This subfunction records the session with the computer system (all output and input) in the specified document.

Name is the name of a document. Folder is up to 30 alphanumeric characters and N is up to 6 digits. Specify the document using the /FOLDER and /NUMBER qualifiers. If no document is specified, then the default document is used (the selected document on the Record Session Menu). If you only specify the name of the document, Communications Control uses the first document with the name it finds in your File Cabinet.

If the document title or the folder you specify is not unique, Communications Control searches for the first document or folder in your File Cabinet that begins with the letters you specify.

If the document you specify does not exist, and you specify a folder (with the /FOLDER qualifier), Communications Control creates a new document.

To specify a name with double quotes, use a pair of double quotes. For example, the word ""title"" has double quotes around it.

Specify the /APPEND qualifier if you want Communications Control to append to text already in the document. Otherwise, Communications Control overwrites the text already in the document.

Communications Control only stops recording the session on this computer system or computer system you subsequently connect to if you specify the STOP RECORDING command or if you leave ALL-IN-1.

SEND[/NOEOL] "text" [WAIT[/TIME = n "text"]]

This subfunction sends a message to the computer system and (optionally) waits for the computer system to send a string back. If you do not want Communications Control to send a carriage return after the text, specify the /NOEOL qualifier. The text is any character string enclosed in double quotes, and the optional WAIT keyword is another CXP subfunction that you can call by itself. The optional /TIME qualifier specifies time, in hundredths of seconds, to wait before WAIT signals an error.

You can also call SEND within the AT subfunction.

SEND/DOCUMENT[/FOLDER = "folder"][/NUMBER = N] ["name"][/[NO]INTERACTIVE]

SEND/DOCUMENT sends File-Cabinet document specified by name to the computer system. You can further identify the document with the /FOLDER and /NUMBER qualifiers.

You typically call this subfunction after the SEND subfunction to prepare the remote computer system to receive a document. For example, if you have established a CXP connection with a remote node, the following function calls create NEWFILE.TXT there:

```
_<CXP SEND "CREATE NEWFILE.TXT"  ! Send the VMS CREATE command
                                 ! to computer system
_<CXP SEND/DOCUMENT ""Memo""     ! Send the File-Cabinet document
_<CXP SEND "<EOF_>"              ! Terminate input with _^Z
```

All send_strings specified by AT subfunction calls are sent at the specified points.

SEND/DOCUMENT/VMS[/[NO]INTERACTIVE] "vms_file_name"

This subfunction works like the previous SEND/DOCUMENT command, but it uses a VMS file and not a file in the ALL-IN-1 File Cabinet.

SET [/[NO]VERIFY] [/[NO]DISPLAY] [/[NO]INPUT]

This subfunction defines certain conditions during the processing of a Control Document. You must specify at least one qualifier. The default is:

SET /NOVERIFY /DISPLAY /INPUT

STOP RECORDING

This subfunction stops recording the session on the computer system and stops Communications Control from recording any subsequent sessions on other computer systems you connect to.

WAIT [/TIME = n] "text"

This subfunction causes Communications to wait until the contents of the string are returned by the computer system before proceeding. You must enclose the string in quotes. The optional /TIME qualifier specifies time, in hundredths of seconds, to wait before WAIT signals an error.

You can also call WAIT from within the SEND or AT commands.

The *User's Reference* covers the CXP subfunctions in detail. The *System Manager's Guide* explains the system-management aspects of Communications CXP control.

12.3.4 CXP Subfunctions that Differ from Control Document Commands

The following Communications Control commands are only available in CXP subfunctions and not as Control Document commands:

- CONNECT/[NO]INTERACTIVE

  The Control Document command CONNECT is available in Function Mode with the addition of the /[NO]INTERACTIVE qualifier. /INTERACTIVE is the default.

  Specify the /NOINTERACTIVE qualifier if you want to connect to the specified computer system without entering Terminal Emulation Mode. Communications Control makes the connection and leaves you in ALL-IN-1 in Function Mode.

- CONNECT/DEVICE/[NO]INTERACTIVE

  The Control Document command CONNECT/DEVICE is available in Function Mode with the addition of the /[NO]INTERACTIVE qualifier. /INTERACTIVE is the default.

  Specify the /NOINTERACTIVE qualifier if you want to connect to the specified computer system without entering Terminal Emulation Mode. Communications Control makes the connection and leaves you in ALL-IN-1 in Function Mode.

- SEND/DOCUMENT/[NO]INTERACTIVE

  The Control Document command SEND/DOCUMENT is available in Function Mode with the addition of the /[NO]INTERACTIVE qualifier. /INTERACTIVE is the default.

  Specify the /NOINTERACTIVE qualifier if you want to return to Function Mode on ALL-IN-1 after the document has been sent.

- SEND/DOCUMENT/VMS[NO]INTERACTIVE filename

  The Control Document command SEND/DOCUMENT/VMS is available in Function Mode with the addition of the /[NO]INTERACTIVE qualifier. /INTERACTIVE is the default.

  Specify the /NOINTERACTIVE qualifier if you want to return to Function Mode on ALL-IN-1 after the document has been sent.

- PROCESS
  Format:
  PROCESS[/FOLDER = FDR][/NUMBER = N][/TEST][/[NO]INTERACTIVE] [TITLE]

  The subfunction PROCESS in not available as a Control Document command. It causes Communications Control to process the Control Document specified with /FOLDER, /NUMBER and title. If no folder, number, or title are specified then the default Control Document is used. The default Control Document is the currently selected document on the Communications Control menu (held in symbol $CXP__CONTROL__DOCUMENT). If a document is specified then this document becomes the default Control Document and $CXP__CONTROL__DOCUMENT is updated accordingly.

  Specify the /TEST qualifier if you only want to check the syntax of the Control Document.

  If you do not want to remain in Terminal Emulation Mode after processing the Control Document, use the /NOINTERACTIVE qualifier.

If you specify the /TEST qualifier, you cannot specify the /[NO]INTERACTIVE qualifier.

- PROCESS/VMS[/TEST][/NO]INTERACTIVE] filename

  Causes Communications Control to process the VMS file specified.

See the *User's Reference* for details on Control Document commands.

12.3.5 Using Communication Control Commands

The system manager can use Control Document commands as part of the definition of a computer system, to specify the wake-up sequence of the computer system. If this is done, the user does not need to know the series of keystrokes that cause an initial response from the computer system when the connection is attempted. The wake-up sequence can also be used to drive unsupported modems. See the *System Manager's Guide* for more details on wake-up commands.

The following table is a summary of how and where you can use Communication Control Commands.

Command	Control Document	Subfunction	Wake-up
CONNECT	Y	Y	N
CONNECT/[NO]INTERACTIVE	N	Y	N
CONNECT/DEVICE	Y	Y	N
CONNECT/DEVICE/[NO]INTERACTIVE	N	Y	N
SEND	Y	Y	Y
SEND/DOCUMENT	Y	Y	Y
SEND/DOCUMENT/[NOINTERACTIVE	N	Y	N
RECORD	Y	Y	Y
STOP RECORDING	Y	Y	Y
WAIT	Y	Y	Y
PAUSE	Y	Y	Y
DISPLAY	Y	Y	Y
SET	Y	Y	Y

(continued)

Command	Control Document	Subfunction	Wake-up
AT	Y	Y	Y
EXIT	Y	N	Y
DISCONNECT	Y	Y	Y
PROCESS	N	Y	N

Where Y = YES and N = NO.

NOTE: As with all interactive function calls, you must have the PRVCMD User Profile privilege before calling CXP subfunctions interactively.

12.3.6 Example of Customizing Communications Control

Assume the users on your system want to connect to one particular computer system frequently. You can customize the Communications menu, adding an option that refers specifically to this computer system and modifying Named Data and the Communications Database so that this option invokes one or more CXP subfunctions successfully. For example, the Communications menu could contain the option:

CSM Connect to STOCK MARKET

where STOCK MARKET is a computer system.

To make this option work:

1. Set up a computer system profile entry for STOCK MARKET. For details of setting up computer systems see the *System Manager's Guide.*
2. Add the option to the main Communications menu, form COM in form library OA$LIB:OAFORM.FLB.
3. Add the following information in the Named Data form for menu COM:

Named Data

```
n Name CSM
CLEAR\CXP CONNECT/INTERACTIVE "STOCK MARKET"\CXP DISCONNECT
```

When you replace form COM in OAFORM and reopen this library, selecting CSM results in the following actions:

1 The CLEAR command clears the screen.

2 The CXP CONNECT "STOCK MARKET" command connects to the computer system STOCK MARKET using the information from the computer system profile for STOCK MARKET. The computer system profile can include Control Document commands that specify the initial wake-up sequences required by the computer system.

3 The INTERACTIVE qualifier on the CONNECT command places the user in Terminal Emulation Mode.

4 The CXP DISCONNECT command is optional and automatically disconnects the user from the computer system when the user presses CTRL and the interrupt character.

5 If you do not include a DISCONNECT command, the connection to the computer system is not broken when the user presses CTRL and enters the interrupt character.

13

Time Management

13.1 Introduction

Time Management is a closely integrated ALL-IN-1 application that derives most of its functionality from the CALENDAR function. This chapter gives an overview of the application and describes the CALENDAR subfunctions.

13.2 Organization

Time Management consists of

- The user's CALENDAR file, OAUSER:username.A1CAL
- The system meeting file, OA$DATA:MEETING.DAT
- The system attendee file, OA$DATA:ATTENDEE.DAT
- The system CALENDAR access file, OA$DATA:CALACCESS.DAT
- The user's Action Items file, OAUSER:ACTITEM.DAT
- The CALENDAR function and its subfunctions
- The forms that provide the Time Management user interface
- Special CALENDAR data sets

In addition, the system User Profile, OA$DATA:PROFILE.DAT, contains a description of each user's work week which is used by several CALENDAR subfunctions.

MEETING.DAT, ATTENDEE.DAT, and CALACCESS.DAT are installed automatically in the directory associated with OA$DATA. Username.A1CAL and ACTITEM.DAT are copied to a user's directory when that user is created. For users created prior to Version 2.1, the CALENDAR file is called username.CAL. Version 2.2 supports CALENDAR files that have either extension.

13.2.1 Time Management Activities

In Time Management a menu displays that allows you to enter one of the following options:

- Action Items — Items to do by a certain time. No checks are made for conflicts with other activities. Action items must be closed out manually.
- CALENDAR Appointments — Appointments are known only to you. They are scheduled within the constraints of other CALENDAR activities during your work week.
- CALENDAR Meetings — Meetings with other ALL-IN-1 users. They are scheduled within the constraints of other CALENDAR activities during your work week and the work week of all attendees.
- Reminders — Items to do on a certain date. Similar to action items, but reminders are one-day ticklers.
- To-Do List — Items to do without a set date. This list contains reminders without a date.

13.2.2 Subfunctions and the Current CALENDAR Date/Time

The current CALENDAR date, by default, is the current system date; it is reset with the CALENDAR SET DATE subfunction.

For example, to set the current CALENDAR date, you could interactively enter:

```
<CALENDAR SET DATE 15-AUG-1986
```

If you do this from the choice field of the main Time Management menu, the new date then displays in the calendar graphic.

The CALENDAR DISPLAY subfunctions use the current date and time to set current meetings and appointments. A current meeting or appointment is the first one scheduled for the current date/time.

13.2.3 CALENDAR Files

The individual CALENDAR file, OAUSER:username.A1CAL, is an indexed RMS file that contains a record for every meeting or appointment scheduled. The file's primary key is a field containing the starting date and time of the meeting or appointment. The secondary key is a field containing a meeting pointer: if a record entry is a meeting, the meeting pointer contains the username of the meeting scheduler and the starting date/time of the meeting.

The meeting pointer corresponds to the primary keys of the two system files, OA$DATA:MEETING.DAT and OA$DATA:ATTENDEE.DAT. Each MEETING.DAT record contains scheduling and resource information about a meeting. The meeting attendees are recorded in ATTENDEE.DAT, with one record per attendee. (Duplicate primary keys are allowed in ATTENDEE.DAT.)

Each user's OA$DATA:PROFILE.DAT record contains fields that describe the calendar workweek and other calendar information.

13.3 CALENDAR Subfunctions

Many CALENDAR subfunctions act upon the current appointment or meeting. Local symbol #CAL_APPT_KEY identifies the current appointment and local symbol #CAL_MEETING_POINTER identifies the current meeting. The DISPLAY subfunction sets both of these symbols, as well as other local symbols that identify current CALENDAR settings.

13.3.1 CALENDAR ADD_ATTENDEE

Syntax: CALENDAR ADD_ATTENDEE

This subfunction adds attendees to the current meeting, which the symbol #CAL_MEETING_KEY identifies.

PARAMETERS:
None

INPUT:
#CAL_MEETING_POINTER
Names in CAL$SCROLL_ATTENDEES

OUTPUT:
Attendees receive a mail message to inform them of the meeting.

13.3.2 CALENDAR CANCEL APPOINTMENT

Syntax: CALENDAR CANCEL APPOINTMENT

This subfunction cancels the current appointment, which the symbol #CAL_APPT_KEY identifies.

PARAMETERS:
None

INPUT:
#CAL_APPT_KEY
OA$TM_OWNER

OUTPUT:
None

13.3.3 CALENDAR CANCEL MEETING

Syntax: CALENDAR CANCEL MEETING

This subfunction cancels the current meeting, which the symbol #CAL_MEETING_POINTER identifies.

PARAMETERS:
None

INPUT:
#CAL_MEETING_POINTER
OA$TM_OWNER

OUTPUT:
Attendees receive a mail message stating that the meeting is canceled.

13.3.4 CALENDAR CLEAR ATTENDEES

Syntax: CALENDAR CLEAR ATTENDEES

This subfunction removes any names that are currently in the CAL$SCROLL_ATTENDEES data set.

PARAMETERS:
None

INPUT:
None

OUTPUT:
CAL$SCROLL_ATTENDEES is empty.

13.3.5 CALENDAR CONVERT APPOINTMENTS

Syntax: CALENDAR CONVERT APPOINTMENTS

This subfunction converts the appointments from Version 1.X format to Version 2.0 format, and creates a new username.CAL file. You must create an ACTITEM.DAT file before using this function.

PARAMETERS:
None

INPUT:
Version 1.X username.CAL file
Version 2.0 ACTITEM.DAT file

OUTPUT:
Version 2.0 username.CAL file

13.3.6 CALENDAR CONVERT ACTION_ITEMS

Syntax: CALENDAR CONVERT ACTION_ITEMS

This subfunction converts the Version 1.X Action Items to Version 2.0 format, and creates a new ACTITEM.DAT file.

PARAMETERS:
None

INPUT:
Version 1.X ACTITEM.DAT file

OUTPUT:
Version 2.0 ACTITEM.DAT file

13.3.7 CALENDAR CONVERT MEETINGS

Syntax: CALENDAR CONVERT MEETINGS

This subfunction converts the Version 1.X meeting records to Version 2.0 format. However, before this function executes, all users on the system must have their personal calendar files converted.

PARAMETERS:
None

INPUT:
Version 1.X username.CAL files
Version 2.0 username.CAL files

OUTPUT:
Version 2.0 username.CAL files are populated.
Version 2.0 OA$DATA:ATTENDEE.DAT file
Version 2.0 OA$DATA:MEETING.DAT file

13.3.8 CALENDAR DISPLAY ANSWER

Syntax: CALENDAR DISPLAY ANSWER

This subfunction checks only those meetings which the user is attending and makes the next unanswered meeting current.

PARAMETERS:
None

INPUT:
OA$TM_OWNER
OA$TM_DATE

OUTPUT:
#CAL_MEETING_POINTER
#CAL_MTNG_WHEN
#CAL_MTNG_STIME
#CAL_MTNG_ETIME
#CAL_MTNG_PRIORITY
#CAL_MTNG_LOC
#CAL_MTNG_DESC
#CAL_MTNG_CONFID
#CAL_MTNG_EXTRA
CAL$SCROLL_ATTENDEES contains the attendee list for the meeting.

NOTE: The CALENDAR DISPLAY subfunctions do not actually display an answer, an appointment, a meeting, and so on. Instead, the subfunction makes the specified item current so that it can be displayed.

13.3.9 CALENDAR DISPLAY APPOINTMENT

Syntax: CALENDAR DISPLAY APPOINTMENT

This subfunction selects an appointment based on the date and time supplied. When an appointment is found, it loads the symbols with the values for the selected appointment. If a user supplies a date and time, a search is made for the first appointment which starts at that date and time. But, if the time is blank, the subfunction selects the first appointment on the specified date.

PARAMETERS:
None

INPUT:
#CAL_APPT_WHEN
#CAL_APPT_STIME
OA$TM_OWNER

OUTPUT:
#CAL_APPT_KEY
#CAL_APPT_WHEN
#CAL_APPT_STIME
#CAL_APPT_ETIME
#CAL_APPT_DESC
#CAL_APPT_CONFID
#CAL_APPT_XTRA

13.3.10 CALENDAR DISPLAY GRAPH

Syntax: CALENDAR DISPLAY GRAPH

This subfunction creates a Time of Day graph for a seven day period. The profile setting determines the position of the current calendar date in the graph.

The graph displays a 12-hour time-period which centers around the user's work day. Each slot in the graph represents 15 minutes, and an asterisk in a time slot indicates a scheduled event. The graph only marks appointments and meetings.

PARAMETERS:
None

INPUT:
USER PROFILE (call day position, start hour, end hour)
OA$TM_OWNER
OA$TM_DATE

OUTPUT:
OA$TM_GRAPH1
OA$TM_GRAPH2
OA$TM_GRAPH3
OA$TM_GRAPH4
OA$TM_GRAPH5
OA$TM_GRAPH6
OA$TM_GRAPH7
OA$TM_DAY1
OA$TM_DAY2
OA$TM_DAY3
OA$TM_DAY4
OA$TM_DAY5
OA$TM_DAY6
OA$TM_DAY7

13.3.11 CALENDAR DISPLAY MEETING

Syntax: CALENDAR DISPLAY MEETING

This subfunction selects a meeting based on the date and time supplied, and loads the symbols with the values for the selected meeting. If the user supplies a date and time, the subfunction searches for the first meeting which starts at that date and time. But, if the time is blank, the subfunction selects the first meeting on the specified date.

The subfunction does not check to determine if the user is the meeting scheduler or an attendee.

PARAMETERS:
None

INPUT:
$CAL_MTNG_WHEN
$CAL_MTNG_STIME
OA$TM_OWNER

OUTPUT:
#CAL_MEETING_POINTER
#CAL_MTNG_WHEN
#CAL_MTNG_STIME
#CAL_MTNG_ETIME
#CAL_MTNG_PRIORITY
#CAL_MTNG_LOC
#CAL_MTNG_DESC
#CAL_MTNG_CONFID
#CAL_MTNG_EXTRA
CAL$SCROLL_ATTENDEES contains the attendee list for the meeting.

13.3.12 CALENDAR DISPLAY MONTH

Syntax: CALENDAR DISPLAY MONTH

This subfunction creates a calendar for the current month, and brackets (<xx>) the current date. An "a" follows the date if an appointment is scheduled for a specific day, while an "m" follows the date if a meeting is scheduled. If there is an action item or reminder, an "r" follows the date.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER

OUTPUT:
OA$TM_MONTH1
OA$TM_MONTH2
OA$TM_MONTH3
OA$TM_MONTH4
OA$TM_MONTH5
OA$TM_MONTH6
OA$TM_MONTH7
OA$TM_MONTH_DISPLAY
OA$TM_YEAR_DISPLAY

13.3.13 CALENDAR DISPLAY NEXT APPOINTMENT

Syntax: CALENDAR DISPLAY NEXT APPOINTMENT

This subfunction selects the appointment which follows the one previously displayed.

The CALENDAR DISPLAY APPOINTMENT function must be executed prior to this subfunction.

PARAMETERS:
None

INPUT:
OA$TM_OWNER

OUTPUT:
#CAL_APPT_KEY
#CAL_APPT_WHEN
#CAL_APPT_STIME
#CAL_APPT_ETIME
#CAL_APPT_DESC
#CAL_APPT_CONFID
#CAL_APPT_EXTRA

13.3.14 CALENDAR DISPLAY NEXT MEETING

Syntax: CALENDAR DISPLAY NEXT MEETING

This subfunction selects the meeting which follows the one previously displayed. The user may be the meeting scheduler or an attendee.

The CALENDAR DISPLAY MEETING function must be executed prior to this subfunction.

PARAMETERS:
None

INPUT:
OA$TM_OWNER

OUTPUT:
#CAL_MEETING_POINTER
#CAL_MTNG_WHEN
#CAL_MTNG_STIME
#CAL_MTNG_ETIME
#CAL_MTNG_PRIORITY
#CAL_MTNG_LOC
#CAL_MTNG_DESC
#CAL_MTNG_CONFID
#CAL_MTNG_EXTRA
CAL$SCROLL_ATTENDEES contains the attendee list for the meeting.

13.3.15 CALENDAR DISPLAY NEXT REPLY

Syntax: CALENDAR DISPLAY NEXT REPLY

This subfunction selects the next meeting at which the user is an attendee.

THE CALENDAR DISPLAY REPLY function must be executed prior to this subfunction.

PARAMETERS:
None

INPUT:
OA$TM_OWNER

OUTPUT:
#CAL_MEETING_POINTER
#CAL_MTNG_WHEN
#CAL_MTNG_STIME
#CAL_MTNG_ETIME
#CAL_MTNG_PRIORITY
#CAL_MTNG_LOC
#CAL_MTNG_DESC
#CAL_MTNG_CONFID
#CAL_MTNG_EXTRA
CAL$SCROLL_ATTENDEES contains the reply information for the user.

13.3.16 CALENDAR DISPLAY REPLY

Syntax: CALENDAR DISPLAY REPLY

This subfunction selects only those meetings which the user is attending. If the user specifies a date and time, the subfunction selects the first meeting at that date and time. But, if the time is blank, it selects the first meeting on the specified date.

PARAMETERS:
None

INPUT:
#CAL_MTNG_WHEN
#CAL_MTNG_STIME
OA$TM_OWNER

OUTPUT:
#CAL_MEETING_POINTER
#CAL_MTNG_WHEN
#CAL_MTNG_STIME
#CAL_MTNG_ETIME
#CAL_MTNG_PRIORITY
#CAL_MTNG_LOC
#CAL_MTNG_DESC
#CAL_MTNG_CONFID
#CAL_MTNG_EXTRA
CAL$SCROLL_ATTENDEES contains the reply information for the user.

13.3.17 CALENDAR INITIALIZE FILES

Syntax: CALENDAR INITIALIZE FILES

This subfunction sets up the default symbols for use by the other subfunctions. It also loads the calendar fields from the User Profile into memory for later use by the Calendar subfunctions.

PARAMETERS:
None

INPUT:
OA$USER

OUTPUT:
OA$TM_OWNER
OA$TM_DATE
OA$TM_DAY
OA$TM_GRAPH_TITLE

13.3.18 CALENDAR INITIALIZE MONTH

Syntax: CALENDAR INITIALIZE MONTH

This subfunction creates a calendar for the current month and brackets (<xx>) the current calendar date.

PARAMETERS:
None

INPUT:
OA$TM_DATE

OUTPUT:
OA$TM_WEEK1
OA$TM_WEEK2
OA$TM_WEEK3
OA$TM_WEEK4
OA$TM_WEEK5
OA$TM_WEEK6
OA$TM_MONTH
OA$TM_YEAR

13.3.19 CALENDAR MAIL_TO_ATTENDEES

Syntax: CALENDAR MAIL_TO_ATTENDEES

This subfunction allows the scheduler of a meeting to send mail to all the attendees of that meeting. It creates a mail message with the names in CAL$SCROLL_ATTENDEES as the addressees, and passes the subject of the mail message to the routine. When the header is created, the subfunction places the user in the editor to add the text to the message. Note that this subfunction may not send the actual message.

PARAMETERS:
None

INPUT:
CAL$SCROLL_ATTENDEES
#CAL_SUBJECT

OUTPUT:
A mail message addressed to the attendees.

13.3.20 CALENDAR MAKE_ATTENDEE_SYMBOLS

Syntax: CALENDAR MAKE_ATTENDEE_SYMBOLS [parameter list]

This subfunction assigns values to local symbols needed to create a reply record and to delete a meeting. The parameter list contains the information which the subfunction loads into the symbols. Typically, multinode/mail uses this subfunction, but it can also be used to allow a command level interface to the other functions.

PARAMETERS:
The subfunction requires all of the parameters, therefore, you must insert place holders if any parameters are blank.

p1 - the meeting pointer
p2 - the comment made by the attendee
p3 - the attendee's status
p4 - the attendee's response
p5 - the attendee's name

INPUT:
Parameter list

OUTPUT:
#CAL_MEETING_POINTER (p1)
#CAL_ATTN_COMMENT (p2)
#CAL_ATTN_STATUS (p3)
#CAL_ATTN_RESPONSE (p4)
#CAL_ATTENDEE (p5)

13.3.21 CALENDAR MAKE_MEETING_SYMBOLS

Syntax: CALENDAR MAKE_MEETING_SYMBOLS [parameter list]

This subfunction assigns values to the symbols which are needed to create/reschedule a meeting. The parameter list contains the information which the subfunction loads into the symbols. Typically, multinode/mail calendar uses this subfunction, but it may also be used to allow a command level interface to the meeting functions.

PARAMETERS:
This subfunction requires all of the parameters, therefore, you must insert place holders if any of the parameters are blank.

p1 — the username of the meeting scheduler
p2 — the description of the meeting
p3 — the location of the meeting
p4 — the date of the meeting
p5 — the starting time of the meeting
p6 — the ending time of the meeting
p7 — not used (reserved)
p8 — the priority of the meeting
p9 — the meeting pointer
p10 — indicator of meeting confidentiality
p11 — customization information for the meeting
p12 — previous meeting pointer if rescheduled

INPUT:
Parameter list

OUTPUT:
#CAL_MTNG_SCHEDULER (p1)
#CAL_MTNG_DESC (p2)
#CAL_MTNG_LOC (p3)
#CAL_MTNG_WHEN (p4)
#CAL_MTNG_STIME (p5)
#CAL_MTNG_ETIME (p6)
#CAL_MTNG_PRIORITY (p8)
#CAL_MEETING_POINTER (p9)
#CAL_MTNG_CONFID (p10)
#CAL_MTNG_EXTRA (p11)

13.3.22 CALENDAR PRINT ALL

Syntax: CALENDAR PRINT ALL

This subfunction creates a file called CALENDAR.TMP which displays the appointments, action items, reminders, and meetings for the current calendar date. It uses the format generated by CAL$SCROLL_BOTH.

The subfunction does not print the file.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER

OUTPUT:
CALENDAR.TMP file

13.3.23 CALENDAR PRINT CONFLICTS

Syntax: CALENDAR PRINT CONFLICTS

This subfunction creates a file called CALENDAR.TMP which displays the meetings and appointments that conflict with the current calendar date. It uses the format generated by CAL$SCROLL_CONFLICTS.

The subfunction does not print the file.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER

OUTPUT:
CALENDAR.TMP file

13.3.24 CALENDAR PRINT CUSTOM

Syntax: CALENDAR PRINT CUSTOM

This subfunction creates a file called CALENDAR.TMP which displays the appointments and meetings for the current calendar date. It uses the format generated by CAL$SCROLL_CUSTOM.

The subfunction does not print the file.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER

OUTPUT:
CALENDAR.TMP file

13.3.24A CALENDAR PRINT DAY_REMIND

Syntax: CALENDAR PRINT DAY_REMIND

This subfunction creates a file called CALENDAR.TMP which displays the reminders and action items for the current calendar date. It uses the format generated by CAL$SCROLL_DAY_REM.

The subfunction does not print the file.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER

OUTPUT:
CALENDAR.TMP file

13.3.25 CALENDAR PRINT DAY_SCHEDULE

Syntax: CALENDAR PRINT DAY_SCHEDULE

This subfunction creates a file called CALENDAR.TMP which displays the appointments and meetings for the current calendar date. It uses the format generated by CAL$SCROLL_DAY.

The subfunction does not print the file.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER

OUTPUT:
CALENDAR.TMP file

13.3.26 CALENDAR PRINT MONTH

Syntax: CALENDAR PRINT MONTH

This subfunction creates a file called CALENDAR.TMP which displays the events that are scheduled for the current month.

The subfunction does not print the file.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER

OUTPUT:
CALENDAR.TMP file
All symbols which CALENDAR DISPLAY MONTH sets up

13.3.26A CALENDAR PRINT TODO

Syntax: CALENDAR PRINT TODO

This subfunction creates a file called CALENDAR.TMP which displays the todo list for the current calendar date. It uses the format generated by CAL$SCROLL_DAY_TODO.

The subfunction does not print the file.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER

OUTPUT:
CALENDAR.TMP file

13.3.27 CALENDAR PRINT TWOCAL

Syntax: CALENDAR PRINT TWOCAL

This subfunction creates a file called CALENDAR.TMP which displays the appointments and meetings for the current CALENDAR user and a second user which a local symbol identifies. The subfunction uses the format generated by CAL$SCROLL_TWOCAL.

The subfunction does not print the file.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER
#CAL_SECOND

OUTPUT:
CALENDAR.TMP file

13.3.28 CALENDAR PRINT WEEK_REMINDERS

Syntax: CALENDAR PRINT WEEK_REMINDERS

This subfunction creates a file called CALENDAR.TMP which displays the reminders for the current calendar week. It uses the format generated by the CAL$SCROLL_WEEK_REM.

The subfunction does not print the file.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER

OUTPUT:
CALENDAR.TMP file

13.3.29 CALENDAR PRINT WEEK_SCHEDULE

Syntax: CALENDAR PRINT WEEK_SCHEDULE

This subfunction creates a file called CALENDAR.TMP which displays the appointments and meetings for the current calendar week. Each day is broken into time slots with the size of each slot being determined by the User Profile. The subfunction uses the format generated by CAL$SCROLL_WEEK_SCH.

The subfunction does not print the file.

PARAMETERS:
None

INPUT:
OA$TM_DATE
OA$TM_OWNER

OUTPUT:
CALENDAR.TMP file

13.3.30 CALENDAR PURGE

Syntax: CALENDAR PURGE

This subfunction removes from the file username.A1CAL all meetings and appointments which occur prior to the specified date.

PARAMETERS:
None

INPUT:
OA$TM_OWNER
#CAL_PURGE_DATE

OUTPUT:
None

13.3.31 CALENDAR REMOVE_ATTENDEES

Syntax: CALENDAR REMOVE_ATTENDEES

This subfunction removes from the current meeting, the list of attendees which the symbol CAL$SCROLL_ATTENDEES identifies.

PARAMETERS:
None

INPUT:
OA$TM_OWNER
#CAL_MEETING_POINTER
CAL$SCROLL_ATTENDEES must contain a valid attendee list.

OUTPUT:
Attendees receive a mail message stating that their attendance is no longer required.

13.3.32 CALENDAR REPLY_TO_MEETING

Syntax: CALENDAR REPLY_TO_MEETING [parameter]

This subfunction allows the user to create a reply to the current meeting request. If a parameter is supplied, the subfunction updates the information for the specified attendee name; otherwise, it updates the information for the current calendar owner.

PARAMETERS:
The username of the attendee who is being asked to reply. This may be an ALL-IN-1 symbol.

INPUT:
OA$TM_OWNER
#CAL_ATTN_COMMENT
#CAL_ATTN_REPONSE
#CAL_ATTN_STATUS
#CAL_MEETING_POINTER

OUTPUT:
Unless the calendar owner is also the meeting scheduler, the subfunction sends a mail message reply to the scheduler.

13.3.33 CALENDAR RESCHEDULE APPOINTMENT

Syntax: CALENDAR RESCHEDULE APPOINTMENT

This subfunction cancels the appointment specified by #CAL_APPT_KEY and reschedules it for a new time.

PARAMETERS:
None

INPUT:
OA$TM_OWNER
#CAL_APPT_KEY
#CAL_APPT_WHEN
#CAL_APPT_STIME
#CAL_APPT_ETIME
#CAL_APPT_DESC
#CAL_APPT_CONFID
#CAL_APPT_EXTRA

OUTPUT:
#CAL_APPT_KEY

13.3.34 CALENDAR RESCHEDULE MEETING

Syntax: CALENDAR RESCHEDULE MEETING

This subfunction cancels the meeting specified by #CAL_MEETING_POINTER and reschedules it for a new time.

PARAMETERS:
None

INPUT:
#CAL_MEETING_POINTER
#CAL_MTNG_SCHEDULER
#CAL_MTNG_WHEN
#CAL_MTNG_STIME
#CAL_MTNG_ETIME
#CAL_MTNG_LOC
#CAL_MTNG_DESC
#CAL_MTNG_PRIORITY
#CAL_MTNG_CONFID
#CAL_MTNG_EXTRA

OUTPUT:
#CAL_MEETING_POINTER
A mail message is sent to attendees to notify them that the meeting has been rescheduled.

13.3.35 CALENDAR SCAN

Syntax: CALENDAR SCAN

This subfunction displays a graph of the meeting scheduler's work day and divides the work day into 15-minute increments. The name of each attendee is listed and a star (*) appears for each 15 minute interval that the attendee is available for a meeting. The graph display stops when all of an attendee's free time is listed, or, it stops on a day that the scheduler specifies.

The subfunction works for up to 36 attendees.

PARAMETERS:
None

INPUT:
Attendee-name.A1CAL

OUTPUT:
A 96 character string that consists of concatenated NBS dates

13.3.36 CALENDAR SCHEDULE APPOINTMENT

Syntax: CALENDAR SCHEDULE APPOINTMENT

This subfunction schedules an appointment at a specified time and date. If another appointment is scheduled for the same time, the user is notified of the conflict.

PARAMETERS:
None

INPUT:
OA$TM_OWNER
#CAL_APPT_WHEN
#CAL_APPT_STIME
#CAL_APPT_ETIME
#CAL_APPT_DESC
#CAL_APPT_CONFID
#CAL_APPT_EXTRA

OUTPUT:
#CAL_APPT_KEY

13.3.37 CALENDAR SCHEDULE MEETING

Syntax: CALENDAR SCHEDULE MEETING

This subfunction schedules a meeting in the user's calendar, as well as in the system-wide meeting and attendee files. If a conflict exists the subfunction notifies users of the conflict and prompts them to determine if they wish to continue. Users are permitted to create multiple meetings which start at the same time.

ALL-IN-1 assumes that the meeting scheduler is also an attendee. If a scheduler adds their name to the attendee list, the error message, `Scheduler already assumed to be an attendee,` displays. The meeting is created, but the scheduler does not receive a meeting notice in the mail.

If you SET FORWARD on your Electronic Messaging menu to forward mail messages to another address, any messages about scheduled meetings are *not* forwarded.

PARAMETERS:
None

INPUT:
OA$TM_OWNER
#CAL_MTNG_SCHEDULER
#CAL_MTNG_WHEN
#CAL_MTNG_STIME
#CAL_MTNG_ETIME
#CAL_MTNG_LOC
#CAL_MTNG_DESC
#CAL_MTNG_PRIORITY
#CAL_MTNG_CONFID
#CAL_MTNG_EXTRA

OUTPUT:
#CAL_MEETING_POINTER
A mail message is sent to attendees notifying them that the meeting has been scheduled.

13.3.38 CALENDAR SET ALTERNATE_USER

Syntax: CALENDAR SET ALTERNATE_USER [parameter]

This subfunction changes the current calendar owner. The user must have profile privilege, schedule access to the alternate user's calendar, and VMS protection access to execute this function successfully.

PARAMETERS:
A valid ALL-IN-1 user name

INPUT:
OA$TM_OWNER
Profile set owner privilege
Calendar access file

OUTPUT:
OA$TM_OWNER

13.3.39 CALENDAR SET DATE

Syntax: CALENDAR SET DATE [parameter]

This subfunction changes the current calendar date.

PARAMETERS:
A valid date command. See the list which follows:

Parameter	Example	Meaning
±[n]D	+2D, -4D	Move forward or back by n days; default is 1
±[n]W	+W, -2W	Move forward or back by n weeks
±[n]M	+3M, -M	Move forward or back by n months
symbol	"12-SEP-1984" "12-SEP-84" "12-SEP"	Move to the specified date
YESTERDAY	YESTERDAY	Move to day before current system date
TODAY	TODAY	Move to today's system date
TOMORROW	TOMORROW	Move to day after current system date
EOM	EOM	Move to end of month

NOTE: Relative values (+M, -2D, and so on) are always relative to the current CALENDAR date, not to the current system date.

Multiple values can be stacked. For example:

```
<CAL SET DATE TODAY +D +W
```

This combination sets the calendar date to a week from tomorrow.

```
<CAL SET DATE +D +W
```

sets the calendar date to a week and a day after the current calendar date. Other examples follow:

```
<CAL SET DATE '15-AUG-1983'

<CAL SET DATE OA$DATE_FULL
```

INPUT:
OA$TM_DATE

OUTPUT:
OA$TM_DATE

13.3.40 CALENDAR SET NEWORK OFF

Syntax: CALENDAR SET NETWORK OFF

This subfunction turns off an internal flag in the present computer system. It is the opposite of the CALENDAR SET NETWORK ON subfunction described below.

PARAMETERS:
None

INPUT:
None

OUTPUT:
None

13.3.41 CALENDAR SET NETWORK ON

Syntax: CALENDAR SET NETWORK ON

This subfunction turns on an internal flag in the present computer system. The flag indicates that a user on a remote system has caused a function to be called in the present system. For example: If a remote user on computer system B wishes to schedule a meeting with a user on computer system A, an internal flag must be turned on in system A. The subfunction,

CALENDAR SET NETWORK ON, accomplishes this, and signals that system A should not send mail messages to the meeting attendees. Computer system B does this task.

PARAMETERS:
None

INPUT:
None

OUTPUT:
None

13.3.42 CALENDAR VERSION

Syntax: CALENDAR VERSION

This subfunction displays the current version of CALENDAR.

PARAMETERS:
None

INPUT:
None

OUTPUT:
Line 24 on the terminal displays a message.

13.4 Calendar File Layouts

13.4.1 User File Layout

The user's calendar file is an indexed RMS file that contains a record for every meeting, and appointment scheduled. The file's primary key is a field containing the starting date and time of the meeting, or appointment. If a record entry is a meeting, the meeting pointer contains the username of the meeting scheduler and the starting date and time of the meeting.

The meeting pointer corresponds to the primary keys of the two system files, OA$DATA:MEETING.DAT and OA$DATA:ATTENDEE.DAT. Each MEETING.DAT record contains scheduling and resource information about

a meeting. The meeting attendees are recorded in ATTENDEE.DAT, which contains one record per attendee. (Duplicate primary keys are allowed in ATTENDEE.DAT.)

A description of the fields in the file username.A1CAL follows:

File Name: username.A1CAL

Field Name	Length	Description
START_DATE	16	The starting date and time of a meeting or appointment in the following format:YYYYMMDDHHMMnnn, where nnn is a sequence number
END_DATE	15	The ending date and time of the meeting or appointment in NBS format: YYYYMMDDHHMM
MEETING_POINTER	46	The primary key of the associated meeting file record which contains additional information for meeting entries; the first 30 characters are the Profile username of the scheduler and the remaining 16 bytes are the date/time of the meeting as it appears in the start date/time field
TYPE	1	A = Appointment M = Meeting
SCHEDULER	1	Y = This is the scheduler N = This is not the scheduler
RESERVED	1	
CONFIDENTIAL	1	Flag: 1 = this appointment is confidential; 0 or blank = this appointment is not confidential
RESERVED	1	
DESCRIPTION	60	Description or purpose of meeting or appointment
FILLER	33	For customization

13.4.2 Meeting File Layout

File Name: OA$DATA:MEETING.DAT

Field Name	Length	Description
MEETING_POINTER	46	The primary key: 30 characters contain the scheduler's name and 16 characters contain the start date and time in the format: YYYYMMDDHHMMnnn
END_TIME	15	The ending date and time in NBS format
MEETING_PURPOSE	60	Description of the meeting
LOCATION	30	Location of the meeting
RESERVED	2	
PRIORITY	2	Defined by user
RESERVED	1	
RESOURCES	30	To tie into a resource scheduling system
FROM_ADDRESS	40	Address of the scheduler
FILLER	50	Customization information

13.4.3 Attendee File Layout

File Name: OA$DATA:ATTENDEE.DAT

Field Name	Length	Description
MEETING_POINTER	46	The primary key: 30 characters contain the scheduler's name and 16 characters contain the start date and time in the format: YYYYMMDDHHMMnnn
ATTENDEE_NAME	30	Name of the attendee for this record
CONFLICT_FLAG	1	Flag: Y = there is a conflict; N = there is not a conflict
RESPONSE	3	May be "Yes","No", or " "
COMMENT	40	Any comment by the user
TRAVEL	16	Travel time for the meeting
SCAN	96	Information to perform a scan
FILLER	19	Additional information for customizing

13.5 CALENDAR Data Sets

See Appendix C for a listing of the CALENDAR data sets

14

The Help and Message Facilities

14.1 Introduction

ALL-IN-1 provides support to both new and experienced users through an on-line Help facility. Help is an implicit option in the system: whenever assistance is needed with a general concept or with a particular form, field, or menu option, calling the HELP function displays context-sensitive information.

The Message facility provides a similar service by displaying one-line messages and by storing messages in a message buffer whenever a user has erred or made a potentially ambiguous entry. In addition, HELP is available on error messages.

14.2 Help Facility

The Help facility consists of the HELP function and a VMS help library containing help text.

14.2.1 The HELP Function

Gold H is associated in the Named Data of form DEFAULT with the HELP function. Thus, pressing Gold H wherever keypad definitions are recognized calls HELP.

HELP can also be called interactively by entering <HELP . . . . in a menu choice field. This function is also valid wherever function calls are allowed.

When HELP is called, it:

1 Opens the help library

2 Assembles a list of keywords that index help-library text modules

3 Calls VMS LIBRARIAN-utility routines to return text based on the keywords

4 Outputs the text through one of three special help forms

14.2.1.1 HELP Context Checking. HELP assembles the keyword list as follows:

1 If HELP is called with parameters, they are treated as Help-library keywords. These keywords are used as the lookup keys into the Help library.

2 When no explicit parameters are supplied, Help looks at the current form (or, more correctly, the form that was current before the special Help form displayed), for context information:

   a If the /HELP form qualifier is present, its arguments are treated as help keywords, overriding any other default context information.

   b If /HELP is not present and if the form is not a menu, then the keyword list is built as:

   ```
   form-name current-field-name
   ```

   where current-field-name is the name of the field that users are in when they call HELP.

   c If /HELP is not present, the form is a menu, and the user is in the CHOICE field, then HELP uses the contents, if any, of the CHOICE field as follows:

   ```
   form-name choice-field-contents
   ```

The choice-field-contents may consist of one or more additional keywords. For example, if the current form were APPLMEN, and if the user enters SEL CURRENT_ITEM in the choice field and then presses Gold H, then HELP searches the default Help library for a topic corresponding to the keyword list APPLMEN SEL CURRENT_ITEM. There are some exceptions to this rule, as will be discussed next.

If the form is a menu and the user is in some field other than CHOICE, then HELP builds the keyword list as if for a nonmenu.

3 If the user calls HELP from a prompt, then HELP looks at the contents of special symbol OA$PROMPT_HELP. If the symbol is nonblank, its contents are treated as a help keyword list, and the Help library is searched using those keywords. If the symbol is blank, HELP builds a keyword list using:

form-name PROMPT

where form-name is the name of the form that is current when the prompt displays, and PROMPT is a fixed keyword that corresponds to a help topic under that form name.

4 If the user is in the editor (current form is OA$EDIT), and presses Gold H, a keypad graphic displays that corresponds to the current editor style. Help is provided on the function keys that are defined for that editor. This keypad graphic is actually an argument form that calls HELP from its Named Data. The effect is the same as if the argument form were called directly from outside the editor.

If HELP is called from a nonblank menu choice field (and no /HELP qualifier is present), the contents of the field are compared to the following combinations before automatically providing help on form-name choice-field-contents:

< anything

If an angle bracket (<) is the first item in the choice field, HELP creates the keyword list as follows:

FUNCTIONS anything

where FUNCTIONS corresponds to a level-1 help topic in the default help library.

$ anything

If a dollar sign ($) is the first item in the choice field, HELP creates the keyword list as follows:

OA$DCL anything

where OA$DCL corresponds to a level-1 help topic in the default help library.

For example, entering <GET in the CHOICE field of the Main Menu and then pressing Gold H displays help on the GET function, indexed by the keywords FUNCTIONS GET.

In all cases, if a given keyword list does not correspond to an existing topic in the help library, HELP trims off the last keyword and tries again. The process continues until the keyword list corresponds to an existing topic or until the list contains one word, and that word does not correspond to a valid topic. In that case, the default help topic, GETTING_HELP, displays.

This backing-off strategy differs slightly when the user has entered something in the CHOICE field and pressed Gold H. For example, entering EM BOGUS BOGUS in the CHOICE field of the Main Menu and then pressing Gold H causes HELP to build the keyword list MAIN EM BOGUS BOGUS. The search for a topic that corresponds to these keywords fails until the keyword list is trimmed to MAIN EM, and Help displays on the EM option.

In contrast, entering MAIN in the CHOICE field of the Electronic Messaging menu and then pressing Gold H causes HELP to build the keyword list EM MAIN, a topic path that fails (unless you have customized the Help library and/or the Electronic Messaging menu). Since options like MAIN are valid at any menu, HELP trims the keyword list differently when the CHOICE field contains information and provides help on the Main Menu instead of the Electronic Messaging menu.

HELP tries to find a match in the HELP library as follows when the CHOICE field contains information when Gold H is pressed:

1 Build the keyword list as form-name CHOICE-field-contents.

2 As long as there is no match in the library, trim off the last keyword until just form-name is left.

3 Instead of trying form-name, start again with CHOICE-field-contents and trim down as long as there is no match in the library.

4 If CHOICE-field-contents by itself did match anything in the library, then replace the form-name and try to get help on it.

14.2.1.2 The HELP Forms. When HELP is called, it displays one of two Help forms — OA$HELP_TOP or OA$HELP_BOTTOM — depending on the cursor position. If the cursor is in the bottom half of the screen, OA$HELP_TOP, an overlay that covers the first 10 lines of the screen, displays. If the cursor is in the top half of the screen, OA$HELP_BOTTOM, an overlay that covers the last 10 lines of the screen, displays. These two help forms are memory resident and are the special HELP form .TYPE.

Since help on keyboard layouts is best presented as graphic information, which does not lend itself to a partial overlay, several CALC forms provide keyboard-style help. The names of these forms have the general format:

```
edit-style_keyboard-type_terminal-type
```

Edit-style corresponds to EDT, WPS, WPL, FMS, and so on. Keyboard-type is either a VT100 or LK201. The VT100 keyboard is represented by a 0 and the LK201 keyboard by a 1. Terminal-type is either advanced video (1) or hard copy (0).

Legal keyboard help form names are as follows:

```
WPS_00   !WPS style editing, VT100, no AVO
FMS_01   !FMS forms editor, VT100, with AVO
EDT_10   !EDT Editor, LK201, no AVO
WPS_11   !WPS style editing, LK201, with AVO
```

If an editor does not support hard copy mode, the last digit (terminal type) remains off. For example:

```
WPL_0    !WPS-PLUS VT100 with AVO
WPL_1    !WPS-PLUS LK201 with AVO
```

When you call HELP from form OA$EDIT (that is, while in the default editor), ALL-IN-1 dispatches a command like:

```
FORM editstyle + keyboard-type + terminal-type
```

For example, for editor style EDT with a VT100 with advanced video the string is:

```
FORM EDT__00
```

If the user enters a defined editor key ALL-IN-1 dispatches a string that looks like:

```
HELP 'KEYBOARD ' OA$PREV__FORM:3:0 {key sequence}
```

If the user's default editor is EDT and the user enters GOLD__T, the command string looks like:

```
HELP KEYBOARD EDT GOLD__T
```

14.2.2 The Help Library

VMS Help libraries are collections of text modules that are indexed by a keyword and arranged by number in a hierarchical tree. A level-1 Help topic, or module, is the highest, or most general, level of help information available. Each level-1 module may have any number of level-2 topics under it, each level-2 topic may have any number of level-3 topics under it, and so on. ALL-IN-1 recognizes up to 8 levels of help.

ALL-IN-1 form names always correspond to level-1 help topics. Enterable fields on a given form always correspond to level-2 topics under that level-1 form name. For example, you are on argument form ALPHA in field OMEGA. You press Gold H. HELP looks at the form name (ALPHA) and the field name (OMEGA) and builds a keyword list (ALPHA OMEGA), where the first keyword, ALPHA corresponds to a level-1 help topic in the library and OMEGA corresponds to a level-2 entry under that topic.

The help module for a nonmenu thus has the following general format:

BEGINNING OF HELP EXAMPLE 1.

```
1 FMS-form-name
Descriptive Title

Help text describing the form and what it does. . . . . .

2 FMS-field-name
Descriptive Title

Help text describing the field and what kind of input is expected...
```

2 FMS-field-name
Descriptive Title

Help text describing the field and what kind of input is expected. . .

END OF HELP EXAMPLE 1.

and so on, where each enterable field in the form is described as a level-2 help topic. For example, the following is the help module in the default help library, OA$LIB:OAHELP.HLB, for entry form FORMAT:

BEGINNING OF HELP EXAMPLE 2.

1 FORMAT
Format Master Entry Form

Use this entry form to maintain the Format Master file, which contains information about the various formats used by Document Processing and Program Development when processing a File Cabinet file.

To leave this form without modifying the file, press EXIT SCREEN. To complete an entry press RETURN. To navigate through the fields, press TAB and/or BACKSPACE.

2 FORMAT
Format Master Name Field

This is the first field on the Format Master entry form, which allows you to maintain information about the different formats used by Document Processing and Program Development. This is also the key field to the Format Master file, and ALL-IN-1 validates your entry into this field against that file. Use the CHOICE keys (Gold L or Gold S key sequences) to view a list of all existing formats.

The contents of this field correspond to a document format: you can optionally specify any format when creating a document or program module. For example, the RUNOFF format indicates that a document has been created as a RUNOFF source file; and when you want to output the document (either to the screen or to the printer), ALL-IN-1 processes that file with RUNOFF first.

Similarly, you would give a module written in a specific programming language the format that corresponds to that language processor.

When you create a document with a format of NONE, ALL-IN-1 does no additional processing. For example, you do not need to process a mail message before you print it, so all mail messages have the format NONE.

=GENERAL RECOG

2 DESC
Format Description Field

You can optionally enter a description of the format in this field.

2 DEFAULT_FILE
Default File Extension Field

This field contains the default file extension associated with the format source file. For example, the RUNOFF format default extension is RNO; the WPS-PLUS default extension is WPL.

ALL-IN-1 often uses the default file extension to determine the internal structure of a file, so this is an important piece of information.

NOTE: The default file extension must correspond to the data set category; help on data set categories is available as an additional topic.

=FORMAT DSAB

2 OBJECTEXT
Object File Default Extension Field

This field contains the default file extension associated with the format object file. The object file is the output file that ALL-IN-1 creates based on the processing instructions associated with a given format. For example, the RUNOFF format default extension is .MEM, which is the file extension that RUNOFF, by default, gives to formatted documents.

2 DSAB
Data-Set Category Field

ALL-IN-1 treats text files as data sets. A text data set is one or more RMS sequential files and information about the contents of those files that ALL-IN-1 maintains while you are processing those files. Treating text files as data sets allows ALL-IN-1 to provide a full range of text processing features for all files, regardless of their internal structure. This field contains a keyword that corresponds to a text data-set category.

A text data-set category is simply a way of grouping files that have the same internal structure. The following are valid data-set categories:

- ASCII — all sequential files in ASCII format
- WPS-PLUS — all sequential files in WPS-PLUS format
- DECdx — all sequential files that have been transferred from a DIGITAL WPS-8 Word Processing system such as a DECmate

2 TEMPLATEDOC
Template Field

2 EDITFUNC
Edit Function Field

2 LISTFUNC
List Function Field

2 PRINTFUNC
Print Function Field

2 STDFORMAT
Standard Formatting Flag Field

2 COMPILE
VMS Compile-Command Field

END OF HELP EXAMPLE 2.

As covered in the previous section, HELP uses form and field names as keywords and tries to match them against corresponding names in the Help library. The =GENERAL RECOG entries at the end of some topics are cross-references.

Since HELP builds the keyword list differently for menus, Help-library text modules for menus are built differently than those for nonmenus. The level-2 topics under menu names are menu options. As with nonmenus, the names of any enterable fields on the menu should also correspond to level-2 help topics; but since the CHOICE field is usually the only enterable field on a menu, the help module usually consists of any explicit (or hidden) options available through that menu.

The help module for a menu thus has the following general format:

BEGINNING OF HELP EXAMPLE 3.

```
1 FMS-form-name
Descriptive title of form

Help text describing the menu and what it does. . .

2 Menu-option-code
Descriptive title of option

Help text describing the menu option and what it does. . .

2 Menu-option-code
Descriptive title of option

Help text describing the menu option and what it does. . .

. . . .
```

END OF HELP EXAMPLE 3.

and so on, where each option, explicit or hidden, is described as a level-2 help topic. For example, the following is the help module in the default help library, OA$LIB:OAHELP.HLB, for the Main Menu:

BEGINNING OF HELP EXAMPLE 4.

```
1 MAIN
The Main Menu Form

You use the Main Menu form as the center of the ALL-IN-1 system. From
the Main Menu you enter into the subsystem options contained in
ALL-IN-1. When you are in the Main Menu, the number of messages
waiting to be read in your inbox is displayed at the top of the form.

To leave the ALL-IN-1 program, type EX and press the RETURN key. To
complete an option selection, press the RETURN key.

For example, to enter the Electronic Messaging subsystem, type EM and
press RETURN.
```

=HELP
=KEYBOARD FMS

2 WP
The Word Processing Subsystem Option

You use the Word Processing (WP) subsystem for document creation, revision, and formatting. This option allows you to perform your daily paperwork requirements such as memos, letters, reports, proposals, and detailed specifications.

=WP
=KEYBOARD FMS

2 EM
The Electronic Mail Subsystem Option

You use the Electronic Messaging (EM) subsystem to prepare, send, and receive mail messages from or to anyone on the ALL-IN-1 system

=EM
=GENERAL MAIL

2 DM
The Desk Management Subsystem Option

You use the Desk Management subsystem to handle your daily activities, such as Calendar Management, Action Item handling, Phone Directory Management, File Cabinet Maintenance, and so forth.

=DM
=KEYBOARD FMS

2 IM
The Information Management Subsystem Option

You use the Information Management subsystem to help handle the needs of information inquiry, data presentation, modeling, and data processing activities.

=IM
=KEYBOARD FMS

2 BI
The Business Information Subsystem Option

You use the Business Information subsystem to store the applications your department uses to help provide information and resources to help you in your function. The functions available from this menu are specific to your business or profession.

=BI
=KEYBOARD FMS

2 PD
The Program Development Subsystem Option

You use the Program Development subsystem to create, modify, and test programs and forms.

=PD
=KEYBOARD FMS

2 PS
The Profession Specific Subsystem Option

You use the Profession Specific subsystem to tailor and hold applications and resources that are specific to your individual needs. This subsystem allows you to add your own options without affecting other users of the system.

=PS
=KEYBOARD FMS

2 US
The User Setup Option

You use the User Setup option to specify a number of operating conditions that affect the way ALL-IN-1 interacts with you.

=KEYBOARD FMS

2 SM
The System Manager Subsystem Option

You use the System Manager option (SM) to allow you to enter the System Manager subsystem.

NOTE: This option does not show on the menu form and is valid only to users with system manager privileges.

=SM
=KEYBOARD FMS

2 TR
The CBI Training Option

You use the CBIs to take a Computer Based Instruction (CBI) course that teaches you the basic functions of ALL-IN-1.

2 EX
The Exit Workstation Option

You use the Exit Workstation option to leave ALL-IN-1. This does not log you out of VAX/VMS.

2 LO
The Leave Workstation Option

You use the Leave Workstation option to leave ALL-IN-1 and log out of VAX/VMS.

END OF HELP EXAMPLE 4.

Note that some of the options (US, EX, and so on) do not appear on the screen display for form MAIN. These are hidden options, defined either in the Named Data of form DEFAULT or in the Named Data of /MORE-linked form MAIN2 and thus recognized as valid options for form MAIN.

In the event that a user presses Gold H from the choice field of a menu, the level-1 Help topic displays, and the user gets help on the form itself, as opposed to any fields or options. For this reason, if a menu is the top-level form in a subsystem (such as WP, EM), the level-1 help topic for that menu should give a brief overview of what the subsystem is and what it does as a whole, as well as what the options are.

14.2.2.1 Additional Topics and Cross-References. When you read an entire help topic, you have the option of viewing a list of additional topics. This list is built automatically by HELP by:

- Searching under the current topic for topics at the next level down
- Searching the text of the current topic for an equal sign (=) in the left margin, which is interpreted as a cross-reference

Cross-references may be literal keywords, an ALL-IN-1 symbol, a symbol list or a keyword list.

For example, if you are in the CHOICE field of form MAIN and press Gold H, the following displays:

```
Current Form: MAIN                                    Field: CHOICE

The Main Menu is the center of the ALL-IN-1 system. From the Main Menu,
you can access the ALL-IN-1 subsystems by choosing from the list of
options.

To select an option, enter the two or three letter abbreviation and
press RETURN. To leave ALL-IN-1, enter EX and press RETURN.
```

and a prompt appears on the bottom of the screen:

```
Press RETURN to continue or EXIT SCREEN to exit
```

Pressing RETURN displays the following:

```
Current Form: MAIN                                    Field: CHOICE

For example, to enter the Electronic Messaging subsystem:

        1 Enter EM.

        2 Press RETURN.

ALL-IN-1 displays the Electronic Messaging menu.
```

and a prompt appears on the bottom of the screen:

```
Press RETURN to continue or EXIT SCREEN to exit
```

Pressing RETURN displays the following:

```
Current Form: MAIN                                    Field: CHOICE

In addition to the list of options, the Main Menu reminds you about

        • the number of mail messages received but not read

        • the number of Action Items pending

        • the number of Calendar activities scheduled for the current
          day
```

and a prompt appears on the bottom of the screen:

```
Press RETURN to continue or EXIT SCREEN to exit
```

Pressing RETURN causes a new prompt to appear at the bottom of the screen:

```
For a list of related topics, press RETURN; press EXIT SCREEN to exit
```

Pressing RETURN displays the following:

```
Current Form: MAIN                                    Field: CHOICE

Help is available on the following topics:

1* The Main Menu
2       The Word and Document Processing Subsystem
3       The Electronic Messaging Subsystem
4       The Desk Management Subsystem
5       The Time Management Subsystem
6       The Information Management Subsystem
```

and another prompt:

```
Enter number from the list and press RETURN; just press RETURN for more
topics.
```

The first item in the list is always the topic you just read (in this case, level-1 topic MAIN), and all other entries on the list are either one level down (such as level-2 topics MAIN WP, MAIN EM, MAIN DM, and so on) or cross-references.

As illustrated, when HELP builds the additional-topics list, it does not present the actual help keywords as options. Instead, the first nonblank line under the keyword line displays as the topic. For example, item 2 in the preceding list, The Word and Document Processing Subsystem Option, is the first line in the topic MAIN WP. Therefore, when building help files, the first line after the keyword should always contain a descriptive title.

Cross-referenced topics are treated as subtopics of the topic that cross-references them. For example, the =KEYBOARD FMS entry at the end of the MAIN text is interpreted by HELP as a cross-reference to the level-2 topic FMS in the level-1 module KEYBOARD. As a result of this cross-reference, KEYBOARD FMS is treated as if it were a level-2 topic under MAIN.

Cross-referencing is generally most useful under level-2 menu options since many menu options only call another form. The help text associated with the option itself may explain why you want to choose this option and what it does, but not much about the form that displays when you choose the option. To provide some information about that form, cross-reference the form name under the option that calls it.

Cross-referencing allows you to keep Help libraries as shallow as possible: that is, by cross-referencing frequently, you do not often have to create level-3 (or deeper) topics. This improves performance during HELP function calls.

When you do add below the level-2 topic, it should be to explain a concept in more detail. It should not be to reference another form that seems to fit into the tree as a level-3 topic. Forms are *always* level-1 topics: cross-reference them to call them as additional options. This is done in the MAIN help module list in this chapter, where, for example, level-1 topic WP (corresponding to form WP) is cross-referenced under level-2 topic MAIN WP (corresponding to the WP option on the Main Menu.)

Further guidelines for cross-referencing: if a topic has limited interest and is unlikely to apply to more than one form or option, then create it as a level-3 topic. If a topic is useful in more than one context, then create it as a level-2 topic under the level-1 catchall GENERAL.

Regardless of where you create the topic, it can always be cross-referenced by another topic.

Assume you have selected The Word and Document Processing Option from the list of additional topics for form MAIN. The text for MAIN WP displays, and if you read it through and respond to the prompt for additional topics, a list very similar to the original list redisplays. The additional topics for MAIN are still listed in the same order, but option 2 — The Word and Document Processing Subsystem Option — is marked with an asterisk, indicating that you have already read that topic. Also, a new topic is appended to the end of the list: The Word and Document Processing Subsystem, corresponding to a cross-reference to level-1 topic WP under level-2 topic MAIN WP. If any level-3 topics had been present under MAIN WP, they too would be appended at the end of this list. (Note that the =KEYBOARD FMS cross-reference under MAIN WP is ignored here because the topic was added to the additional topics list by an earlier cross-reference.)

HELP continues to indicate already-read topics with an asterisk and adds to the additional-topics list as long as the user selects an item from the list and requests more Help afterwards.

14.2.2.2 Line-1 Cross-References. When =keyword-text appears as the first line of help text, that entry is treated as a direct reference and not a cross-reference. That is, everything after that first line is ignored, and the text referenced by the keywords displays.

For example, the Help library contains the following entries:

```
1 KEYBOARD
ALL-IN-1 Keyboard Definitions

. . .

2 FMS
Key Definitions for ALL-IN-1 Forms

. . .

3 GOLD_A
=GENERAL KEYDEFS GOLD_A

3 GOLD_B
=GENERAL KEYDEFS GOLD_B

. . . .
```

Subtopic FMS of topic KEYBOARD explains the special-function keys that are defined for ALL-IN-1 forms. However, these definitions also appear in another part of the help library: under subtopic KEYDEFS of topic GENERAL. Instead of repeating the text, it is referenced directly. Thus, the interactive call:

```
<HELP KEYBOARD FMS GOLD_A
```

is equivalent to:

```
<HELP GENERAL KEYDEFS GOLD_A
```

Topics at any level may be cross-referenced directly in this manner.

In summary, every time you use the HELP function, you are instructing it to search the default Help library for a given topic. If you choose to look at additional topics, a list is built from the next-level-down topics and from any cross-references. The original topic always remains additional-topic 1. When you select from the additional-topics list, subtopics of the selected topic are added to the list if you choose to look at additional topics again (and each time subsequently).

14.2.3 Maintaining, Updating, and Switching Between Help Libraries

The default ALL-IN-1 help library, OA$LIB:OAHELP.HLB, is an enhanced VMS help library. All of the help-library features described in the VAX/VMS documentation set are supported, as is the ALL-IN-1 cross-reference feature.

OAHELP consists of over 130 level-1 help modules: one for each shared form, and several others for miscellaneous topics. OAHELP consists of RUNOFF-formatted text; the RUNOFF source text is stored in OA$BUILD:HLPSOURCE.HLB.

14.2.3.1 Updating the Help Libraries. Use the DCL LIBRARIAN Utility to examine and maintain the ALL-IN-1 help library. To examine the existing modules in OAHELP, use the DCL LIBRARY command with the /LIST and /HELP qualifiers from within ALL-IN-1:

```
$ LIB/LIST/HELP OA$LIB:OAHELP
```

To save the list in a file, use a /LIST argument:

```
$ LIB/LIST = HELPTOPIC.TXT/HELP OA$LIB:OAHELP
```

To update an existing module of help text in OAHELP:

1 First extract that module from the unformatted library, OA$BUILD:HLPSOURCE.HLB. The /OUT qualifier can be specified to control where the extracted module is placed and what it is named:

```
$ LIB/EXTRACT = module_name/HELP/OUT = module_name.RNO OA$LIB:HLPSOURCE
```

where module_name corresponds to a level-1 module of help text. Edit the extracted module, and format it using RUNOFF commands. The .page size and .no paging RUNOFF commands should be added at the top of the module, and the .br should be added after the keyword to ensure that it is on a line of its own after formatting:

```
.! Since the size of the scrolled region on forms
.! OA$HELP_TOP and OA$HELP_BOTTOM is a fixed width
.! of 75 lines, make the help-library text that width.
.!
.page size ,75
.!
.! Since no page breaks are necessary for scrolled
.! text (such as help)...
.!
.no paging
.!
.! Specify AUTOPARAGRAPH to reduce the number
.! of RUNOFF commands needed, and ensure that
.! paragraphs begin at left margin
.!
.autopar
.paragraph 0,1
.!
.! Make sure that topic-level/keyword is on line of its own
.!
1 MAIN
.br
The Main Menu Form

Text.....
```

The .br command also ensures that a cross-reference remains on a line of its own after formatting. The comments are provided for clarity and may be omitted.

2 Format the module with RUNOFF:

```
$ RUNOFF module_name
```

3 Replace both the source text and the newly formatted text:

```
$ LIB/HELP OA$LIB:OAHELP module_name.MEM

$ LIB/HELP OA$BUILD:HLPSOURCE module_name.RNO
```

4 The changes are in effect the next time you call the HELP function.

To add a module of help text (to explain a newly created form, for example), create it as a RUNOFF source file. Format it as above, and add the source and formatted text to the help libraries with the same LIBRARY commands used to replace existing modules.

14.2.3.2 Restrictions on Building the Help Library. The following list covers restrictions that are unique to ALL-IN-1 Help libraries as well as those that apply to all Help libraries.

- Level-2 topics and below should always be in alphabetical order. Otherwise, the librarian may misinterpret a keyword that is a proper subset of another keyword, and you may repeat additional topics. For example:

  ```
  2 KEYDEFS
  3 GOLD_DEFAULT
  3 GOLD_D
  ```

  displays the title line from GOLD_DEFAULT twice in the additional topics list.

- When you do a line-1 cross-reference to a topic that is partly valid (that is, first N keyword(s) is/are good, last one(s) is/are bad), you may repeat additional topics.

- The VMS Help facility is designed for DCL commands and their qualifiers, and a / in column 1 of any help module is treated like a numbered subtopic. Thus, you may get unexpected additional topics if your formatted help text contains slashes in the first column.

A similar type of problem occurs when your formatted help text includes numerals that start in the first column. In all cases, the librarian treats these as topic and subtopic indexes and issues an Illegal key error message when you try to insert or replace the module in the Help library.

14.2.3.3 Switching Between Help Libraries. When HELP opens the help library, it first looks in the main-process logical-name tables for OAHELP. If this logical has been assigned (with the LOGICAL function) to some alternate help library, then the logical name is expanded and the LIBRARIAN attempts to open that library. If OAHELP is not a main-process logical, it is interpreted as [ ]OAHELP.HLB: that is, as help library OAHELP.HLB in your current default directory (usually OAUSER). If this library does not exist or cannot be opened, then the default library, OA$LIB:OAHELP.HLB, is opened.

The OAHELP logical allows you to switch between existing help libraries while ALL-IN-1 is running. For example, the following function calls open the DATATRIEVE help library and return the text associated with WOMBAT ADVANCED:

```
<LOGICAL OAHELP, SYS$HELP:DTRHELP.HLB \ HELP WOMBAT ADVANCED
```

The help-library search strategy also allows you to keep and update a personal version of OAHELP.HLB in your default ALL-IN-1 directory.

NOTE: If OAHELP is defined in the subprocess (for example, $ ASSIGN/PROC OAHELP equiv-name at a menu), it is not recognized by HELP. The logical must be defined in the main process. The only way to create a main-process logical name while ALL-IN-1 is running is with the LOGICAL function.

Also, if (as in the example) you create OAHELP and assign it a value with the LOGICAL function, you must also explicitly reassign it with LOGICAL if you want to switch back to another library. For example, specify:

```
< LOGICAL OAHELP,OA$LIB:OAHELP.HLB
```

Calling the LOGICAL function with a null second parameter does not delete a logical, and, in the case of OAHELP, causes an error the next time you call HELP.

14.2.4 Help at a Prompt

When HELP (Gold H) is called from a prompt, ALL-IN-1 first looks at special symbol OA$PROMPT_HELP for a keyword list. If that symbol is empty, then HELP builds the keyword list as form-name PROMPT, where form-name is the form that was current when the prompt was called and PROMPT is a fixed level-2 keyword that can be added to the help library like any other keyword.

You can assign OA$PROMPT_HELP a value with the GET function:

```
<GET OA$PROMPT_HELP = "GENERAL PHELP"
```

Since you can make this assignment dynamically anywhere GET function calls are allowed, you can provide different help for different prompts, or even for the same prompt called at different times.

If OA$PROMPT_HELP is blank, you can obtain help on ALL-IN-1 error messages automatically by pressing Gold H at the prompt associated with special form OA$VIEW_MESSAGES. After you call the PROMPT function, remember to clear the symbol OA$PROMPT_HELP so that the value that it contains is not used for the next PROMPT function call. The following command clears the OA$PROMPT_HELP symbol:

```
<GET OA$PROMPT_HELP = ""
```

14.3 Message Facility

The Message Facility comprises:

- A primary message file, OA$BUILD:OAMESS.MSG, which contains the text of ALL-IN-1 messages
- A message buffer, which accumulates messages generated by ALL-IN-1 and the VMS facilities that it calls
- A signal handler, which traps messages and stores them in the message buffer, and also unwinds from routines which have aborted while allowing the rest of ALL-IN-1 to continue functioning

The message file and the message buffer are explained in the following sections.

14.3.1 The Message File

OA$BUILD:OAMESS.MSG is a standard VMS message file that contains all the messages issued by ALL-IN-1. Messages are grouped by severity and indexed by keyword. The severity of a message indicates whether the facility that generated the message completed its task successfully, and the message keyword can be used as a help keyword to obtain more information.

Messages have the following general syntax:

```
%fac-s-keyword ,text
```

where

%fac — is a facility code (usually OA),

s — is a severity code, either:

W	for warning
I	for informational
F	for fatal
E	for error
S	for success

keyword — is a keyword that indexes the message text in the message file.

text — is the text of the message.

For example:

```
%OA-E-FORMNF, Cannot find form "WO"
```

This error message is indexed in the message file by keyword FORMNF.

14.3.2 Updating the Message File

You may modify the text of ALL-IN-1 messages by editing the message file as an ASCII text file. *Do not alter existing message keywords or severity groupings.* OAMESS.MSG is stored in the ALL-IN-1 OA$BUILD directory.

Format guidelines for message files are covered in detail in the VAX-11 documentation set. Use the DCL MESSAGE command to compile the updated OAMESS.MSG into an object module, OAMESS.OBJ; for example:

```
$ MESSAGE OA$BUILD:OAMESS
```

Use the resulting object module to replace the existing message module in the ALL-IN-1 object library, OALIBR.OLB:

```
$ LIBRARY OA$BUILD:OALIBR OAMESS
```

Relink ALL-IN-1 as follows:

1. Execute the command procedure OA$BUILD:A1LNKDRV.COM:

   ```
   $ @OA$BUILD:A1LNKDRV.COM
   ```

2. Copy the new image to OA$LIB using the command:

   ```
   COPY A1.EXE OA$LIB:A1.EXE
   ```

3. Install the new image as described in the *ALL-IN-1 Installation Guide.*

The new image displays the revised error messages.

14.3.3 Message Severity and the Message Buffer

Each message severity level is translated into a specific ALL-IN-1 action:

- Warning — Message is stored in the message buffer for later display, processing continues.
- Informational — Message is stored in the message buffer for later display, processing continues.
- Fatal — Message is stored in the log file, displayed on the terminal, and ALL-IN-1 aborts.
- Error — Message is stored in the message buffer for later display. The message is also stored in the log file.
- Success — No message is issued, processing continues.

As noted some messages are stored in a message buffer for later display. The message buffer is a storage area for the ALL-IN-1 which is equivalent to VMS linked messages.

When ALL-IN-1 performs a task, it often calls VMS system services and other utilities. If these calls result in errors, each error may generate a message. Several messages of varying severity may accumulate before the user ever sees any indication that something went wrong. The final message — that is the message closest to the ALL-IN-1 user — is the only one that displays on line 24 when the most recently completed task has signaled some type of unsuccessful completion. This final message is the primary message, and any other messages that were generated are stored behind it in the message buffer.

Each time a task generates one or more messages, ALL-IN-1 clears the message buffer and saves the latest group of linked messages. The following special symbols, which ALL-IN-1 updates each time it refills the message buffer, contain the components of the primary message:

OA$MSG__FAC — Facility prefix of primary message

OA$MSG__ID — Message ID of primary message

OA$MSG__SEV — Severity code of primary message, either:

- W — Warning
- I — Informational
- F — Fatal
- E — Error
- S — Success

OA$MSG__TEXT — Text of the primary message

14.3.3.1 Viewing the Message Buffer.

Special VIEW form OA$VIEW__MESSAGES displays the current contents of the message buffer through a scrolled region whenever it is called. This form, which is a short overlay at the bottom of the screen, is associated in the Named Data of form DEFAULT with Gold W.

For example, if you edit or create a form, and there is an error in Named Data, when you try to call that form, one message displays on line 24:

```
Error occurred while form (form_name) was current
```

Pressing Gold W displays the additional messages that were generated along with this primary message. In this case, a Parsing Error message usually underlies the primary message. This secondary message pinpoints the exact location in Named Data where the error occurred.

14.3.4 Help on Error Messages

Although pressing Gold W displays all messages generated at a given time, these single lines of text may not always be self-explanatory. Thus, level-1 help module ERRORS contain many level-2 topics that correspond to ALL-IN-1 error-message keywords.

In most cases, it is necessary to call HELP with explicit parameters to display this help text; for example:

```
< HELP ERRORS FNF
```

returns the help for the FNF (or Form Not Found) error message.

15

The Script Facility

15.1 Overview

The Script facility processes the indirect command files, or scripts, that you specify as parameters of the SCRIPT or DO functions. Scripts are used to create:

- Computer Based Instructions (CBIs)
- User Defined Processes (UDPs)
- Command streams when running ALL-IN-1 in Batch Mode
- System demonstrations
- Installation Verification Procedures (IVPs)

15.2 Scripts

As an interactive system, ALL-IN-1 depends on input to initiate processing. The user usually provides this input as menu choices, terminator keys, key sequences, and so on. Scripts can also provide this input, a kind of "pseudo input" capability for ALL-IN-1.

Scripts are indirect command files, analogous to DCL command procedures. A script often interacts with the user. It may request input from the user which it may filter or verify.

15.2.1 Script Modes

When you invoke a script, you invoke it in SCRIPT Mode or DO Mode. These two modes correspond to the SCRIPT and DO functions. The one you choose depends on how you want to satisfy ALL-IN-1 input requests while the script is active.

A SCRIPT Mode script stands in for the user and provides input to ALL-IN-1 that ALL-IN-1 would normally request the user to supply at his terminal. For example, if you invoke a script with the SCRIPT function and call a form from within that script, ALL-IN-1 first looks to the current script for input when it reaches an enterable field.

A DO Mode script does *not* supply input to ALL-IN-1. In DO Mode the current script is turned off during ALL-IN-1 requests for input; the user must enter all input, with possible exceptions for nested DO scripts. (See the following section on mixing Script Modes.)

15.2.1.1 Mixing Modes. Both SCRIPT and DO are ALL-IN-1 functions, and can be called from within a script, regardless of how that script is invoked. Scripts of different modes can be nested within one another, but the modes remain distinct.

When ALL-IN-1 needs input while a nested SCRIPT script is current, it first looks to the deepest nested SCRIPT script. If that script exits while ALL-IN-1 is still expecting input, control passes to the previous nested SCRIPT script; and, ALL-IN-1 seeks input from it. This process continues until the outer-most script is reached. If ALL-IN-1 is still looking for input when the outermost script exits, it requests it from the user.

When ALL-IN-1 needs input while a nested DO script is current, it first looks to the deepest nested SCRIPT script. If there is no SCRIPT script active, ALL-IN-1 seeks input from the user.

In SCRIPT-invoked scripts, you can specify the .FX directive, which calls a function or list of functions in DO Mode. Examples are given in the .FUNCTION and .FX directives.

15.2.2 Interactive Directives and the Psuedo-Input Buffer

The Script facility updates a pseudo-input buffer every time the user, or a script, has supplied input while a script is active. This buffer is maintained as a special symbol, OA$SCRIPT_PSIB.

OA$SCRIPT_PSIB is assigned a value when:

- You call one of the following interactive directives in a script:

  .GET_FIELD — OA$SCRIPT_PSIB saves input into field.

  .PROMPT — OA$SCRIPT_PSIB saves response to the prompt.

  .SINGLE — OA$SCRIPT_PSIB saves response to the prompt (one character or mnemonic).

  .SINX — OA$SCRIPT_PSIB saves response to the prompt (one character or mnemonic).

- You specify a text line in the script; the Script facility loads OA$SCRIPT_PSIB with the text.
- You provide a parameter with the .PROCESS directive; the Script facility loads OA$SCRIPT_PSIB with the value of the parameter.

Script trims off all leading and trailing spaces and tabs before it saves a value in OA$SCRIPT_PSIB. Special characters are translated into their corresponding mnemonics (see Appendix D), enclosed in script mnemonic delimiter characters, and saved in that form in the input buffer. The terminator is also translated and stored as a mnemonic.

NOTE: The standard American version of ALL-IN-1 uses the brace characters ({ and }) as script mnemonic delimiters. Alternative language versions and customized versions of ALL-IN-1 may use different delimiters. Consult your system manager to find out if your system uses other delimiters besides the brace characters.

The Script facility passes the contents of OA$SCRIPT_PSIB to ALL-IN-1 automatically when it encounters a .PROCESS directive or when there is a text line in the script. Since these two features are invalid in DO Mode, the Script facility never passes OA$SCRIPT_PSIB automatically to ALL-IN-1 from a DO-invoked script. However, as a special symbol, the input buffer is still available to ALL-IN-1 while a DO-invoked script is

active. For example, user input into a field, as requested by .GET_FIELD, can be examined directly by checking the contents of OA$SCRIPT_PSIB, either with another Script directive or with a function called while the script is running.

OA$SCRIPT_PSIB is an updatable special symbol: that is, it can be loaded with any value you wish to assign it. For example:

```
<GET OA$SCRIPT_PSIB = "Coatamundi"
```

loads the specified string in the buffer. As with any other special symbol, you can use OA$SCRIPT_PSIB as a source destination in a GET function call:

```
<GET #LAST_INPUT = OA$SCRIPT_PSIB
```

You may also access the contents of OA$SCRIPT_PSIB through two special symbols: OA$SCRIPT_TEXT and OA$SCRIPT_TERM. These two symbols return the text character string and the terminator character from the pseudo-input buffer, respectively.

15.2.3 Exiting from Scripts

There are several ways to terminate scripts:

- Complete the script by running off the end.
- Use the .EXIT directive to exit from the current script.
- Use the .ABORT directive to exit from the current script and from all levels of script nesting.
- Press Gold X or CTRL/C: like the .ABORT directive, these keys cause the Script facility to unwind from all levels of nesting.

The Script facility defines Gold X and CTRL/C internally. If a script is not active, ALL-IN-1 does not recognize Gold X unless it has been redefined with a keypad directive. You cannot redefine CTRL/C.

When the Script facility encounters an .ABORT, or when you press Gold X or CTRL/C, Script first looks at special symbol OA$SCRIPT_EXIT_HANDLER, which you can load with the name of an ALL-IN-1 function call (typically DO cleanup_script_name) that Script dispatches before returning control to the point of call.

15.2.4 ALL-IN-1 TXLs

15.2.4.1 Definition of a TXL. TXL is a term that is specific to ALL-IN-1. It is a collection of text elements which ALL-IN-1 uses frequently. The collection consists of SCRIPT scripts, DO scripts, and MERGE boilerplates. When ALL-IN-1 starts up, a TXL combines text elements into a structure that is faster to access and more memory efficient than individual on-disk RMS files for each element.

There are three VMS directories that contain the original scripts and boilerplates. The directories are specified in OAGBL.BLI - their definitions, the associated logicals, and their equivalents follow:

Element Type	Logical	Directory
SCRIPT scripts	"OA$SCP"	[ALLIN1.SCP]
DO scripts	"OA$DO"	[ALLIN1.DO]
MERGE boilerplates	"OA$BLP"	[ALLIN1.BLP]

If a logical specified for a directory name is undefined, or if the directory does not exist, or if the directory is empty, the TXL does not contain any elements of that type.

15.2.4.2 Building and Modifying TXLs. If you create new files or change existing files in the TXL directories, you can:

1. Call the OA$TXL_COMPILE function in order to update the TXL.
2. Call the OA$TXL_CLOSE function to explicitly close and reopen a TXL. This ensures that the new TXL is available within the current ALL-IN-1 session. If you do not use the OA$TXL_CLOSE function, the previous TXL remains current until you run ALL-IN-1 again.
3. Compile the TXL using the CTX option available from the System Manager Functions additional options menu. The CTX option calls and executes the following functions in the following order:
   - OA$TXL_CLOSE
   - OA$TXL_REMOVE
   - OA$TXL_COMPILE
   - OA$TXL_INSTALL

The TXL that a session is using can be closed only when its usage count goes to zero. For example, when there are no more active scripts or boilerplates being used from a TXL, the TXL can be closed using the function OA$TXL_CLOSE.

When you create new elements in a TXL directory, you must ensure that the file name of the element you are creating is unique. However, it is permissible and sometimes appropriate to have the same file name across different TXL source directories. For example, the following file names are correct:

FOO.SCP

and

FOO.BLP

15.2.4.3 Search Order for TXL Elements. ALL-IN-1 searches for scripts and boilerplates in the following order:

1. The TXL
2. The user directory
3. OA$LIB
4. OA$DATA

ALL-IN-1 follows this order if you reference a script or boilerplate without a directory specification. For example, if you invoke

```
<SCRIPT ZZZ
```

ALL-IN-1 first searches the TXL, then OA$USER:ZZZ, then OA$LIB:ZZZ and so forth.

In developing TXL elements, you usually copy the elements to your directory and modify them there. If you want to test a new element in the TXL and you do not want to recompile and reinstall the TXL each time you make a change, you can use the OA$TXL_SEARCH_LAST symbol. This symbol acts like a switch and enables/disables the search order for TXL elements. To use the TXL default search order, specify a parameter of 0. To change the search order, specify a parameter of 1. For example:

```
<GET OA$TXL_SEARCH_LAST = 1
```

The parameter 1 specifies that the following search order is used:

1 The user directory

2 OA$LIB

3 OA$DATA

4 The TXL

Using the OA$TXL_SEARCH_LAST symbol allows you to make changes to and test TXL elements, without recompiling the TXL. Other ALL-IN-1 sessions access the previous elements until you recompile.

If you modify the element in its TXL source directory (for example, in OA$BLP or OA$DO), the change is not seen since the default directory search order does not include OA$BLP, OA$DO, or OA$SCP.

See the *System Manager's Guide* for more information on TXLs.

15.2.5 Script-File Format

A generalized script file contains two types of entries: directives and text. Directives have the syntax:

```
.directive [parameter-string]
```

where the required period is the first character in the line, and directive is any valid script directive. The directive may take a string of one or more parameters. For example:

```
.CLEAR 2,10
```

In a SCRIPT Mode file, the Script facility treats all lines that are not directives as text, and in a DO Mode file, treats all lines that are not directives as function calls. Text lines in SCRIPT Mode scripts are sent as they are encountered, to whatever ALL-IN-1 facility is currently requesting input, and that facility usually outputs the text lines on the screen. Thus, text lines immediately follow directives and other text lines that have invoked some ALL-IN-1 facility that is requesting input.

Text lines in SCRIPT Mode scripts may include special characters enclosed in script mnemonic delimiters. Special characters are explicit spaces, TABs, carriage returns, printing characters such as form feeds, keys and key

sequences not on the main keyboard, Gold-key sequences, CTRL-key sequences, and so on. The mnemonics and numeric equivalents of all character sequences are discussed in Appendix D.

For example:

```
The quick brown fox{CR}
```

When it processes a text line, the Script facility ignores all actual carriage returns, TABs, form feeds, and trailing and leading spaces. You can use these characters to enhance the readability of the script file, but you must enclose their mnemonics or numeric equivalents in braces to send them correctly back to ALL-IN-1.

NOTE: To enter a script mnemonic delimiter as an input character, you enter either the numeric equivalent or an acceptable alias mnemonic. For example, for the Left Brace character, you enter either {123} or {LBRACE}. For the Right Brace character, you enter either {125} or {RBRACE}.

Appendix D contains two kinds of entries: primary and alias entries. A primary entry is the mnemonic that is associated with a numeric value such that when the numeric value must be converted to a mnemonic equivalent, the text of the primary entry is used. Thus, {SPACE} is the primary entry in the table for the character whose numeric equivalent is 32.

Alias entries are alternative mnemonics that can be used to enter a numeric equivalent value. {SP} is an alias for {SPACE} (value is 32).

The table of Script mnemonics is a customizable feature of ALL-IN-1. You should check with your system manager to see if there are any differences and/or additions to the table for your site.

The following script file segment illustrates the use of special characters:

```
The quick brown fox{TAB} jumped{32}over the lazy dog{CR}
```

The line includes a TAB and an explicit space ("32" is the numeric equivalent of a space), and it ends with a carriage return. The Script facility sends the following text back to ALL-IN-1:

```
The quick brown fox          jumped over the lazy dog
```

As in the preceding example, a text line or series of lines usually ends with a special character that corresponds to a terminator key or key sequence. If you do not specify a terminator, the Script facility stores the line in OA$SCRIPT__PSIB and appends the next text line (or other input) to it. Since you usually send input to ALL-IN-1 (when ALL-IN-1 requests it) with a terminator such as a TAB or carriage return, a script text line with no terminator often causes unexpected results.

If the text line ends with a space that must be input, the ending space must be explicitly added using the special character format:

```
The quick brown fox{SPACE}
jumped over the lazy dog.{CR}
```

yielding the input line:

```
The quick brown fox jumped over the lazy dog.{CR}
```

If you terminate either a text line or a directive with a hyphen, the Script facility appends the next line to it. For example:

```
.PROMPT 21, -
5, "Press RETURN to re—
enter the CBI lesson."
```

OR

```
The quick bro-
wn fox{TAB}jumped{32}over -
the lazy dog{CR}
```

15.3 Script Directives

Descriptions of the script directives follow.

.ABORT

The .ABORT directive exits from all levels of SCRIPT and DO scripts. Pressing Gold X or CTRL/C while a script is running has the same affect as .ABORT has.

When Script encounters the .ABORT directive, it looks at updatable special symbol OA$SCRIPT__EXIT__HANDLER for a possible script exit-handling routine.

```
.BLOCK
.
.
.
.END_BLOCK
```

The .BLOCK directive delineates a series of text lines which follow it as text to be output as if they were a sequence of .TEXT directives. A .BLOCK block is terminated by the .END_BLOCK directive.

Blank lines are significant inside of .BLOCK blocks.

The syntax is:

```
.BLOCK [line[,column]]
```

which outputs on the screen the series of text lines following it up to the next .END_BLOCK directive. If line is specified, the first text line is output on that line of the screen. If column is specified, all text lines in the block start at that column (in effect, a left margin for the text block).

For example:

```
.BLOCK 20,10
text line
text line
.
.
.
text line
.END_BLOCK
```

puts the text lines on the screen starting at line 20, with each line beginning in column 10.

```
.CHECK_FIELD
```

The .CHECK_FIELD directive checks that the specified field is the current field. Scripts can get out of sync with what is happening in ALL-IN-1, either because the script or the user interface changed. .CHECK_FIELD simply compares the current field name with what is supplied by .CHECK_FIELD, and if they do not match, the script outputs an error message and exits.

The script itself can establish the current field with the .FUNCTION FIELD field_name directive.

The syntax is:

.CHECK_FIELD field-name

For example:

.CHECK_FIELD DOCNAME

checks if the current field known to ALL-IN-1 is DOCNAME. If it is not, the script outputs the error message and exits.

.CHECK_FORM

The .CHECK_FORM directive checks that the specified form is the current form. Scripts can get out of sync with what is happening in ALL-IN-1, either because the script or the user interface changed. .CHECK_FORM simply compares the name of the current or most recently processed form with what is supplied by .CHECK_FORM, and if they do not match, the script exits.

The script itself can establish the current form with the .FUNCTION FORM form_name directive.

The syntax is:

.CHECK_FORM form-name

For example:

.CHECK_FORM WP

checks if the current or most recently processed form known to ALL-IN-1 is WP. If not, the script exits.

.CLEAR

The .CLEAR directive clears the screen. If not otherwise specified, the entire screen is cleared.

The syntax is:

.CLEAR [start-line [,no-lines]]

where the optional parameters are the starting line from which to clear and the number of lines to clear. When omitted, the whole screen is cleared.

For example:

.CLEAR 5,4

clears 4 lines beginning at line 5.

When .CLEAR is called as a function without parameters, it acts like the CLEAR function.

The .CLEAR directive is not meaningful when ALL-IN-1 is running in Hard Copy Mode.

.CURSOR

The .CURSOR directive moves the cursor to the specified position on the screen.

The syntax is:

.CURSOR line,col

where the two required parameters are the line number and the column where the cursor is to be positioned. For example,

.CURSOR 15, 20

positions the cursor on line 15 in column 20.

.CURSOR is not meaningful when ALL-IN-1 is running in Hard Copy Mode.

.CUR_KEEP

The .CUR_KEEP directive activates the CUR_KEEP Mode. In CUR_KEEP Mode, the cursor position and video attributes are saved before the screen is rearranged by an output directive and are restored to the original setting after that directive finishes.

The syntax is:

.CUR_KEEP [OFF]

.CUR_KEEP activates the mode, and .CUR_KEEP [OFF] deactivates it. The following output directives are affected by this mode:

.DISPLAY
.PROMPT
.SINGLE
.TEXT
.WAIT

When CUR_KEEP Mode is active, then .PROMPT, .SINGLE, .TEXT, and .WAIT display text with the last video attributes, if any, specified with .VIDEO_ATT.

This directive is not meaningful when ALL-IN-1 is running in Hard Copy Mode.

.CUR_SAVE

The .CUR_SAVE directive saves the current cursor position and attributes. A .CUR_SAVE replaces any previous .CUR_SAVEs. Other facilities that save the cursor using VT100 ESC sequences override .CUR_SAVE. This directive is not nestable.

.CUR_RESTORE

The .CUR_RESTORE directive restores the current cursor position and attributes saved by the .CUR_SAVE directive. This directive is not nestable.

.DISPLAY

The .DISPLAY directive displays the contents of a text file on the screen. The file may contain VT100 graphics, REGIS, or SIXEL graphics. No check of terminal type is made. The default file name extension is .DRW.

The syntax is:

.DISPLAY [line][,column,]file-name

where the required parameter is the VMS file specification of the file to display. If line is specified, the first text line is output on that line of the screen. If column is specified, all text lines in the file start at that column (in effect forming a text margin).

For example:

```
.DISPLAY 20,10,[GRAPHICS]WOMBAT
```

displays file WOMBAT.DRW (in the GRAPHICS directory) on the terminal, starting at line 20, with each line beginning in column 10.

This directive can be called as a function, but calling the LIST function directly is more efficient.

.EXIT

The .EXIT directive causes the script to exit. If the current script file is not nested in another script, control returns to the point where the SCRIPT function was called. The effect is the same as when [EOF] is detected in the script file.

The syntax is:

```
.EXIT
```

Gold X and CTRL/C in a script also cause an orderly exit from the current script as well as from all levels of nesting. Gold X is only recognized at a script request for input, while CTRL/C can be pressed at any time. However, if CTRL/C is pressed during an input request from the keyboard, ALL-IN-1 completes the input request before exiting.

.EXIT is not meaningful when called singly as a function.

.FUNCTION

The .FUNCTION directive command calls an ALL-IN-1 function or list of functions. The syntax is:

```
.FUNCTION  function[\function...]
```

where function... is one or more ALL-IN-1 functions. The full syntax for function calls is given in Chapter 6.

The SCRIPT and DO functions can be called through .FUNCTION, thus allowing scripts to be nested. For example:

```
.FUNCTION DO INNRSCRPT
```

executes the file INNRSCRPT.SCP with a DO function call. The directives in INNRSCRPT execute in DO Mode, and SCRIPT Mode resumes when INNRSCRPT exits.

If the function is interactive or requires additional input, the current script provides the input in the form of text lines following the .FUNCTION directive.

NOTE: Since text lines in DO-invoked scripts are themselves treated as function calls, .FUNCTION is superfluous in these scripts. In DO-invoked scripts, .FUNCTION OA$HELLO is the same as .FX OA$HELLO and is also the same as OA$HELLO by itself.

.FX

The .FX directive is similar to .FUNCTION, except that .FX hides active script input from the function(s) it invokes. This means that while the .FX function or function list is being executed, any input requested by the function or functions does not come from the current script(s), as is the case with .FUNCTION; but instead comes from the user.

NOTE: Text lines (not starting with a command) are interpreted as function calls in DO Mode scripts. Thus, the .FUNCTION and .FX directives are superfluous in DO scripts. SCRIPT Mode is NOT deactivated during a .FX call from a DO script. This allows you, for example, to call a function with .FX from within a nested DO script, and have the active SCRIPT script (if any) provide the parameters to that function.

The syntax is:

```
.FX function[\function...]
```

where function... is one or more ALL-IN-1 functions. The full syntax for function calls is given in Chapter 6.

You can call the SCRIPT and DO functions through .FX and thus nest scripts. For example:

```
.FX SCRIPT TESTALL\FORM RESULTS\IFEXIT\COMMAND CHECKALL
```

When the Script facility encounters this directive, it first activates or reactivates SCRIPT Mode (if needed) since the first function is a SCRIPT call. When TESTALL exits, SCRIPT Mode is again deactivated for the remaining three function calls. When the function list has finished executing, the original mode is restored.

.FZ

The .FZ directive functions as the .FX directive does except that it hides active DO scripts from the functions that it invokes (the .FX hides SCRIPT scripts).

The syntax is:

```
.FZ function[\function...]
```

where function... is one or more ALL-IN-1 functions. The full syntax for function calls is given in Chapter 6.

This directive is used to prevent a nested DO script (for example, one invoked from a menu option) from continuing back into a stacked DO script (for example, one invoked from a CBI script) when the original script terminates. Typically, .FZ should be used in Named Data for all menu options that involve DO scripts.

For example:

```
Name   Data
C      .FZ FORM CXCRDAT
```

if the form CXCRDAT or its options invoke other DO scripts.

.GET_FIELD

The .GET_FIELD directive places the user in the named field and stores any user input in special symbol OA$SCRIPT_PSIB, replacing the previous contents (if any) of that symbol. .GET_FIELD should only be used when the script is displaying an FMS form and the current (or specified) field is enterable (FMS non display-only).

The syntax is:

```
.GET_FIELD [field-name]
```

where the single, optional parameter is the name of the field to get. If field-name is omitted, the user is placed in the current field.

The Script facility can do several things with the contents of the field:

- Examine with the .JUDGE directive or with an .IF OA$SCRIPT_PSIB... directive.
- Return control to ALL-IN-1 with the .PROCESS directive and let ALL-IN-1 process the input.
- Call a function, with the .FUNCTION or .FX directives, that processes the contents of the field, either directly or indirectly.

The contents of the field are trimmed of trailing blanks, and the terminator is the last character. Thus, a subsequent .JUDGE or .IF examination of the field contents should be done on the exact field contents, plus the terminator.

For example:

```
.GET_FIELD QUERY
```

places the user in form field QUERY, performs FMS field input, and stores any input, plus the terminator, in symbol OA$SCRIPT_PSIB.

.GOTO

The .GOTO directive directs control of the script to the line following a matching .LABEL directive. If the label is not found, the Script facility issues an error and the script exits.

The syntax is:

```
.GOTO label-name
```

where label-name is a literal name identified by a .LABEL directive elsewhere in the script. For example:

```
.GOTO WORK_AROUND
```

transfers control to the following line of the script:

```
.LABEL WORK_AROUND
```

.GOTO is not meaningful when called singly as a function.

.IF

.IF tests the specified conditional expression. If it is true, the THEN clause is executed. If it is false and there is an ELSE clause, the ELSE clause is executed.

The syntax is:

```
.IF expression THEN then-directive [ELSE else-directive]
```

where

expression — is a conditional expression. This expression may range from one ALL-IN-1 symbol (tested for a value of 0 or 1) to a Boolean expression containing references to ALL-IN-1 symbols and data sets.

THEN — is a required keyword, and then-directive is any script directive or ALL-IN-1 function(s) to execute if the expression is evaluated as true.

ELSE — is an optional keyword; when specified, the ELSE else-directive is any script directive or ALL-IN-1 function(s) to execute if the expression is evaluated as false.

Some examples follow.

```
.IF OA$TERMINAL:3 EQS "VT1" THEN .DISPLAY CABINET.VT1 -
ELSE .DISPLAY HARDCAB.TXT
```

This directive tests the value of special symbol OA$TERMINAL to determine whether the user is on a VT1... series terminal. If so, the .DISPLAY directive displays `CABINET.VT1`; otherwise, it displays `HARDCAB.TXT`.

```
.IF #SEL:1 GES "M" AND DEPT EQS "ADMIN" THEN .FUNCT COMMAND
CHECKTIME
```

If the first character of local symbol #SELECT is greater or equal to "M" and field DEPT on the current form contains the value "ADMIN", then the .FUNCTION directive calls the COMMAND function to execute CHECKTIME.COM.

```
.IF OA$STATUS THEN .FX FORM ASKMORE\IFEXIT\DO NEEDMORE
```

OA$STATUS is a special symbol, maintained by a number of ALL-IN-1 facilities, such as the File Cabinet, that contains either 0 or 1 to indicate the status of the last facility action. If it contains 1 here, .FX executes the three functions listed.

NOTE: In Named Data, the .FX directive may dispatch more than one function call. To distinguish between these function calls and any other functions that may be invoked before or after the entire .IF expression, separate each .FX call with a double backslash. This convention is also used when calling multiple functions from within a FOR function.

```
.JUDGE
.
.
.
commands
.
.
.
.END__JUDGE
```

The .JUDGE directive compares the contents of special symbol OA$SCRIPT__PSIB with a text pattern or patterns. All characters are changed to uppercase letters before they are compared. On a match, script processing continues normally. If the comparison fails, control passes to the script line following the next .END__JUDGE directive in the script.

Text patterns can consist of special and permanent ALL-IN-1 symbols and of text-string/special-character combinations enclosed in quotes. When multiple text patterns are specified, only one pattern has to match. Wildcarding can be specified.

The syntax is:

```
.JUDGE text-pattern [,text-pattern...]
.
.
.
commands
.
.
.
.END__JUDGE
```

where text-pattern is a symbol or quoted string, against which to compare. The text-pattern may contain special characters in script mnemonic delimiters, including the following wildcard characters:

{?} Wild-character. These special characters may be placed anywhere in a quoted text pattern and are interpreted as a match-any-character symbol.

{*} Wild-string. This special character may begin a quoted text pattern. It means that only the text and/or special characters that follow it in the text pattern must match the contents of the input buffer.

For example:

```
.JUDGE
"MAIL{CR}","{*}MAIL{CR}",OA$FOLDER,"{*}{TAB}","MAIL{?}{?}{CR}"
```

results in a match if the user inputs:

"Mail", "mail", or "MAIL" (or any other upper/lowercase combination) terminated by a carriage return

OR

An entry of any length that ends with the word "MAIL" and is terminated by a carriage return

OR

An entry that matches the value of special symbol OA$FOLDER

OR

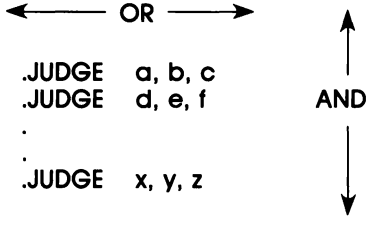
An entry of any length terminated by a TAB

OR

"MAIL" followed by any two nonblank characters and terminated by a carriage return

If none of these options match the user input, control passes to the script line following the next .END_JUDGE command.

More than one .JUDGE directive may be specified before an .END_JUDGE, and they are not processed in the same way as when all the text patterns are specified in one .JUDGE. The user's input is matched against a cross-product combination of the text patterns from each .JUDGE rather than against a long string of all possible inputs. Multiple .JUDGES are evaluated as follows:



For example:

```
.JUDGE
"MAIL{?}","GENERAL{?}","PLAN{?}","{*}{CR}",{*}{TAB}","{*}{ENTER}"
 commands
.END_JUDGE
```

This combination of text patterns result in a match if the user enters:

"mail", "general", or "plan", followed by any terminator; or any entry terminated by either a carriage return, TAB or ENTER.

However, when the text patterns are specified as follows:

```
.JUDGE "MAIL{?}", "GENERAL{?}", "PLAN{?}"
.JUDGE "{*}{CR}", "{*}{TAB}", "{*}{ENTER}"
    commands
.END_JUDGE
```

a match results in only nine cases: when the user inputs either "mail", "general", or "plan" terminated by either a carriage return, TAB or ENTER.

.JUDGE and .END_JUDGE are not valid when called singly as functions.

.LABEL

The .LABEL directive identifies a .GOTO point in the script. The syntax is:

```
.LABEL label-name
```

where label-name is a literal that serves as a destination for one or more .GOTO directives.

.LABEL is not meaningful when called singly as a function.

.PAUSE

The .PAUSE directive causes the script to pause for the specified number of seconds. The .PAUSE directive is most useful in demonstration scripts to allow viewers to read screens before proceeding to another step.

See also the .SET_PAUSE directive, which sets an automatic pause interval for each input step.

The syntax is:

.PAUSE seconds

For example:

.PAUSE 5

freezes the action of the script for 5 seconds.

.PROCESS

The .PROCESS directive causes the script to temporarily return control to ALL-IN-1. It is called when the script has obtained input from the user and wants that input processed by ALL-IN-1.

.PROCESS is only meaningful if the script has been invoked by the SCRIPT function and is not a valid directive in DO scripts. Each text line in a SCRIPT-invoked script is followed by an implicit .PROCESS.

Control returns automatically to the script at the next line in the script.

The syntax is:

.PROCESS [input-symbol]

where input-symbol is an ALL-IN-1 symbol that is translated and stored in OA$SCRIPT_PSIB. This parameter is optional: .PROCESS automatically returns the current contents of OA$SCRIPT_PSIB when it is called, so if input-symbol is omitted, the symbol is simply not updated.

Text lines are followed by an implicit .PROCESS. Since text lines are automatically loaded into OA$SCRIPT_PSIB, the directive:

.PROCESS A line of text{CR}

is equivalent to:

A line of text{CR}

The difference is that .PROCESS can specify symbols. For example:

.PROCESS OA$DATE_FULL

.PROCESS PROFIL.DIRECT[OA$USER] "{CR}"

In the latter example, the contents of user-profile field DIRECT for the current user are concatenated with a carriage return, and the resulting value is sent to ALL-IN-1 in OA$SCRIPT_PSIB. Another example follows.

```
! Get input from field DOCNAME
!
.GET_FIELD DOCNAME
!
! The response has been stored in OA$SCRIPT_PSIB;
! Check the terminator
!
.JUDGE "{*}{TAB}","{*}{CR}"
!
! Reached this point when user has pressed TAB or
! carriage return. Return control to ALL-IN-1 to process
! contents of OA$SCRIPT_PSIB (that is, contents of
! field DOCNAME)
!
.PROCESS
!
! Ensure that the current field is called FOLDER
!
.CHECK_FIELD FOLDER
!
! Pick up at line ".LABEL NEXT_FIELD"
!
.GOTO NEXT_FIELD
.END_JUDGE
!
! Reached this point when user has not pressed TAB or
! carriage return. Check the TERMINATOR for BACKSPACE
!
.JUDGE "{*}{BS}"
!
! Return control to ALL-IN-1 to process contents of
! OA$SCRIPT_PSIB
!
.PROCESS
!
! Ensure that the current field is called BEFORE_DOCNAME
!
.CHECK_FIELD BEFORE_DOCNAME
!
! Pick up at line ".LABEL PREVIOUS_FIELD"
!
.GOTO PREVIOUS_FIELD
.END_JUDGE
```

.PROCESS is not valid when you call it singly as a function.

.PROMPT

The .PROMPT directive displays a specified prompt and waits for user input, terminated by any valid terminator key or key sequence. Any input is stored in special symbol OA$SCRIPT__PSIB and may be analyzed by the .JUDGE, or .IF directives or processed by the .PROCESS directive.

The syntax is:

```
.PROMPT [line[,column],] [prompt-symbol]
```

where

line	is the line on which to prompt. If this optional parameter is omitted, the prompt occurs at line 24 by default.
column	is the column in the line where the prompt begins. If this optional parameter is omitted, the prompt occurs in column 1 of the specified line.
prompt-symbol	may be either a quoted string or a symbol-table symbol that contains the text of the prompt. If this optional parameter is omitted, no prompt displays.

Examples follow.

```
.PROMPT 4, "What do you want to look at now?"
```

displays the quoted text beginning on line 4 and then waits for user input immediately after the prompt.

```
.PROMPT 2,20,$STANDARD__PROMPT
```

displays the string contained in the symbol $STANDARD__PROMPT at line 2, column 20.

Pressing Gold X at a .PROMPT prompt causes the current scripts and any higher-level nested scripts to exit.

.REFRESH

The .REFRESH directive refreshes a specified portion of the screen that was overwritten during script processing — possibly because of the .CLEAR or .DISPLAY directives.

This directive refreshes all lines flagged by any .SET_REFRESH directives specified since the last .REFRESH and includes any lines specified directly as parameters to .REFRESH. The syntax is:

```
.REFRESH [start-line [,no-lines]]
```

where

start-line — is an optional parameter identifying the starting line of the area to refresh.

no-lines — is an optional parameter identifying the number of lines to refresh. If no-lines is omitted, only start-line is refreshed, in addition to any lines flagged by .SET_REFRESH.

If more than one range of lines needs to be refreshed, this directive can be repeated. For example:

```
.REFRESH 1,5
.REFRESH 12
```

This pair of directives refreshes lines 1 through 5 and line 12 (plus any lines already flagged by .SET_REFRESH). However, it is more efficient to call .SET_REFRESH more than once in this manner and then repaint the screen with a single .REFRESH.

.REMARKS

.REMARKS allows comments to be inserted into the script. These comments are ignored when the script is processed.

The syntax is:

```
.REMARKS anything
```

where anything is any text that helps clarify the script file. An exclamation point ("!") in column 1 of a script line also is equivalent to .REMARKS.

For example:

```
.REMARKS Now test for synchronization
.CHECK_FIELD DOCNAME
!
! Continue processing
!
```

NOTE: Totally blank lines are also ignored in scripts, except inside text blocks bounded by the .BLOCK and .END_BLOCK directives. They are equivalent to lines beginning with ! or .REMARKS with no text.

.RESET_CURSOR

SCRIPT .RESET_CURSOR resets cursor and video attributes immediately to their original state.

NOTE: On FMS forms, .RESET_CURSOR takes effect only when ALL-IN-1 processes the next FMS field or form. While in the EDT, WPS or WPS-PLUS editors, the .RESET_CURSOR takes effect immediately.

.SET_PAUSE

The .SET_PAUSE directive sets an automatic pause interval that determines how long the calling facility waits each time after it receives and/or acts upon script input. .SET_PAUSE stays in effect until turned off, whereas .PAUSE is a one-time-only pause after script directives.

The interval can be reset more than once within the script, or it can be set to zero (no delay). ALL-IN-1 initializes the script delay to 0 when it begins running.

The syntax is:

.SET_PAUSE seconds

For example:

```
.PROMPT "Do you want to slow down the script? (Y or N): "
.JUDGE "Y{*}"
.SET_PAUSE 3
.END_JUDGE
```

This sequence prompts the user to make a decision about the speed of the script and sets the delay to 3 if the script is running too fast (useful when checking out a script to see if it is operating properly).

.SET_REFRESH

The .SET_REFRESH command flags a portion of the screen that was overwritten during script processing — possibly because of the .CLEAR or .DISPLAY directives — so that ALL-IN-1 or the .REFRESH directive can

restore it to its original state. If you want to leave script-generated text on part of the screen, do not call .SET_REFRESH for those lines. See also the .REFRESH directive.

The syntax is:

```
.SET_REFRESH [start-line[,no-lines]]
```

where

start-line — is an optional parameter identifying the starting line of the area to flag.

no-lines — is an optional parameter identifying the number of lines to flag for refreshment. If no-lines is omitted, only start-line is flagged.

If you omit both parameters, the entire screen is flagged for refreshment.

If you need to flag more than one range of lines for refreshment, you can repeat .SET_REFRESH, and the effect is cumulative. For example:

```
.SET_REFRESH 1,12
.SET_REFRESH 23
```

This pair of directives flags lines 1 through 12 and line 23 to be refreshed.

.SHOW_FORM

The .SHOW_FORM directive displays an FMS form on the screen. As in the SHOW_FORM function, the form does not become the current form and is not readied for processing by ALL-IN-1. This directive only shows the user what a form looks like.

The syntax is:

```
.SHOW_FORM form-name
```

where form-name is the FMS name of the form to display. For example:

```
.SHOW_FORM MAIN
```

displays form MAIN. If, as in this example, .SHOW_FORM displays a menu, that menu does not become part of the prior menu stack but, instead, is a one-time-only display.

To call a form and interact with it through the script, use ".FUNCTION FORM form-name".

Although you can call .SHOW_FORM as a function, the SHOW_FORM function is more efficient. .SHOW_FORM has no effect in Hard Copy Mode.

.SINGLE

.SINGLE accepts a single character from the user. The character may be a regular character, a blank, or a special terminator key or key sequence. In all other respects, .SINGLE is the same as .PROMPT.

The syntax is:

```
.SINGLE [line[,column],][prompt-symbol]
```

where

line	is the line on which to prompt on. If this optional parameter is omitted, the .SINGLE prompt occurs at the current cursor position.
column	is the column in the line where the prompt begins. If this optional parameter is omitted, the .SINGLE prompt occurs in column 1 of the specified line.
prompt-symbol	may be either a quoted string or a symbol-table symbol that contains the text of the .SINGLE prompt. If this optional parameter is omitted, no prompt displays.

For example:

```
.SINGLE "Press the ENTER key to confirm, EXIT SCREEN to exit"
```

The character or terminator is saved in OA$SCRIPT_PSIB.

Pressing Gold X at a .SINGLE prompt causes the current scripts and any higher-level nested scripts to exit.

.SINX

Like .SINGLE, .SINX accepts any single character or key-sequence from the user. However, .SINX accepts no parameters, displays no prompt, and does not echo the input or change the cursor position or video attributes.

The syntax is:

.SINX

The cursor remains at its current position both before and after the user responds. The character or terminator is saved in OA$SCRIPT__PSIB.

Pressing Gold X at a .SINX prompt causes the current scripts and any higher-level nested scripts to exit.

.TEXT

The .TEXT directive outputs text at a specified location on the screen. The syntax is:

.TEXT [line[,column],]text-symbol

where

line — is the line on which to output the text. If this optional parameter is omitted, the text is output at the current cursor position.

column — is the column in the line where the output begins. If this optional parameter is omitted, the output begins in column 1 of the specified line.

text-symbol — is any ALL-IN-1 symbol representing the text to display.

.TRACE

The .TRACE directive enables script TRACE Mode. This mode is analogous to DCL VERIFY Mode: when TRACE is set, SCRIPT prints out each command just before executing it. The syntax is:

.TRACE [OFF]

Calling .TRACE enables TRACE Mode, and calling .TRACE OFF disables it.

If you enable TRACE Mode in a script, it remains enabled after the script exits. .TRACE OFF must be explicitly specified to turn tracing off.

.VIDEO_ATT

The .VIDEO_ATT command sets a video attribute for other directives that display information. The syntax is:

.VIDEO_ATT { BLINKING | BOLD | REVERSE | UNDERLINE | OFF }

Only one of these optional keywords can be specified, and it overrides the previous setting. .VIDEO_ATT affects the output of the .PROMPT, .TEXT, .TRACE, and .WAIT directives and is only valid while in CUR_KEEP Mode. This mode is set with the .CUR_KEEP directive.

.VIDEO_ATT is only meaningful within the context of a script and cannot be called singly as a function.

.WAIT

The .WAIT directive displays a message and waits for the user to press carriage return before continuing. The syntax is:

.WAIT [line[,column],] [wait-symbol]

where

line — is the line to display the message on. If this optional parameter is omitted, the message displays at the current cursor position.

column — is the column in the line where the message begins. If this optional parameter is omitted, the message displays in column 1 of the specified line.

wait-symbol — may be either a quoted string or a symbol that contains the text of the message. If this optional parameter is omitted, the default message `Press RETURN to continue` displays.

Unlike .PROMPT and other interactive directives, .WAIT does accept input and does not alter OA$SCRIPT_PSIB. This directive simply creates a pause in the script and lets the user decide when to resume.

Pressing Gold X at a .WAIT prompt causes the current scripts and any higher-level nested scripts to exit. When .WAIT is called as a function, it is similar, but not identical to, the WAIT function.

15.4 Script Directives as Functions

Most Script directives can be called as ALL-IN-1 functions and are valid wherever function calls are allowed. These directives complement the control functions documented in Chapter 6. In addition, several of them assign values to OA$SCRIPT_PSIB. The following directives are only meaningful in the context of a script and may not be called singly as functions:

.EXIT
.GOTO
.JUDGE/.END_JUDGE
.LABEL
.REMARKS
.VIDEO_ATT

Any special restrictions on calling other directives as functions are covered under the individual directive.

Single directives can be invoked from Named Data:

```
1 HELLO
.IF OA$USER = = #USER THEN .FX GET HI = "Hi Bert"
```

Menu option HELLO calls Script .IF, which compares the values of two symbols and GETs a message into field HI if the symbols are equal.

Interactively, the syntax is the same:

```
<.IF OA$STATUS THEN .FX .PROMPT "File Cabinet updated; press RETURN"
```

Also, validation qualifiers can specify any single SCRIPT command in a validation expression.

15.5 Sample Script

The following script demonstrates ALL-IN-1 hard copy features.

```
!          *** HARDCOPY.SCP: demo of Hard Copy Mode ***

.TEXT 22,1,"This script demonstrates HARD COPY Mode."
.TEXT 23,1," ********************  "
.TRACE OFF
.SET_PAUSE 1
.TEXT 24,1"Entering HARD COPY Mode using ALL-IN-1 function MODE C"
.PAUSE 1
<MODE C{CR}
<FORM MAIN{CR}
.TEXT "Typing '<TIME{ESC}{ESC}' will first display the form..."
.PAUSE 2
<TIME{ESC}{ESC}
.TEXT "Now a {CR} will accept that value & show TIME"
{CR}
.PAUSE 2
.TEXT "Let's look at the User Profile"
US{CR}
UP{CR}
.TEXT "Tabbing down to STATUS, entering a new value"
{TAB}
{TAB}
{TAB}
script status{TAB}
.TEXT "Tabbing down further and then back up (using BACKSPACE)"
{TAB}
{TAB}
{BS}
{BS}
{BS}
.PROMPT "Now, enter your own status value and press {TAB}"
.JUDGE "{*}{TAB}"
.PROCESS
.END_JUDGE
.TEXT "Pressing {ESC}{ESC} to show the form"
{ESC}{ESC}
.TEXT "Exit the form with {CR}"
{CR}
.TEXT "Confirm with another {CR}"
{CR}
.TEXT "Pressing {ESC}0 twice (hard copy ""EXIT SCREEN"") -
to EXIT back to the MAIN menu"
{ESC}0{ESC 0}
.TEXT "Going back to SCREEN Mode . . ."
<MODE S
.PAUSE 2
{CR}
MAIN{CR}
.EXIT
```

15.6 User Defined Processes

User Defined Processes (UDPs) are scripts that you call with the ALL-IN-1 Gold U response.

15.7 Running ALL-IN-1 Indirectly or in Batch Mode

When you run ALL-IN-1 through a command procedure, you can supply all input in the form of SCRIPT Mode commands. When you run ALL-IN-1 in Batch Mode, you must supply all input in Batch Mode. The simplest way to do this is to integrate a SCRIPT Mode script directly into the command procedure. For example:

```
$ RUN ALLIN1
<TIME{CR}
WP SEL{CR}
{BS}{LF}{TAB}
Anytitle{CR}
{CR}
P{CR}{KEY 0}
{KEY 0}
$EXIT
```

Note that explicit carriage returns (or other terminators) must be included after all input.

15.8 Special Features and Restrictions

15.8.1 Creating a Script with a Script

You can use a script to create a script as long as you exercise caution. A script can invoke an editor. For example, you can write a UDP which invokes the UDP Create option, in the default editor. The script then waits for input from the user or script. At this point the UDP is ready to send text lines to the editor. These lines constitute the UDP script which the UDP is creating.

You *must* use the .PROCESS command to send the text lines to the editor. If you send script commands to the editor (for example, the script commands which will be part of the UDP you are creating) without the .PROCESS command in front of them, they look like commands to be executed immediately and do not appear as commands to send to the editor. An example follows.

```
!Bring up the UDP menu and create a UDP.
UD C{CR}
!Fill in the name of the UDP to create.
!Then ALL-IN-1 invokes the editor.
UDPTEST{CR}
Here are the lines to send to the editor; they become the UDP.
.PROCESS '.TEXT "Hi, a UDP created me!"'
.PROCESS '.PROMPT "Enter your name: "'
.PROCESS '.IF OA$USER NES OA$SCRIPT_TEXT THEN .GOTO NOTSO'
.PROCESS '.PROCESS'
.PROCESS '.EXIT'
.PROCESS '.LABEL NOTSO'
.PROCESS '.TEXT "Sorry, this is not your name"'
.PROCESS '.EXIT'
!Exit the editor; the UDP is created.
{GOLD F}
.EXIT
```

15.8.2 UDPs for the Desk Calculator

Because ALL-IN-1 processes input in the Desk Calculator one character at a time, the field in which information is entered is only 1 character wide and has the autotab attribute set. For Desk Calculator UDPs to emulate the input exactly as ALL-IN-1 processes it, the input must include a {TAB} after each character.

For example, to correctly enter 1+1= you must include a TAB after each character:

```
1 {TAB}
+ {TAB}
1 {TAB}
= {TAB}
```

16

Application Development Tools and Design Guidelines

16.1 Introduction

When designing an application to solve a specific business problem, it often turns out that only a small part of the application addresses the actual problem. Most of the programming effort involves overhead such as gathering user input, opening and closing data files, presenting reports back to the user, and so on.

ALL-IN-1 is designed to minimize the detail work that invariably accompanies any computer programming effort. For most types of applications, the framework is already in place: you can add options to existing ALL-IN-1 menus or build applications around new menus.

Still, you nearly always have to choose one of several paths at all times while developing an ALL-IN-1 application. The various form .TYPES, qualifiers, and functions can be implemented in a variety of ways. This chapter covers how best to use the development tools and features that are available for building applications.

16.1.1 Where to Start

Before you begin writing an ALL-IN-1 application, you should have a thorough understanding of what the system does and how it does it. There are many tools throughout ALL-IN-1 that simplify application development and have the added advantage of presenting new functionality in familiar surroundings.

For example, your organization may need a monthly-report application to keep track of progress in an area. You can create MERGE-function templates and make them available to all users by storing them as shared file-cabinet documents.

You can position the application as an option on the Profession Specific or Business Applications menus. Then, whenever an individual needs to create a report, the tools to do so are available from a menu at or near the top of the ALL-IN-1 menu tree, and the features it incorporates (Edit, Index, and so on) will be familiar to those who use these features in other applications.

The exercises in Appendix A illustrate features that can be put to use in many applications.

Once you have a clear idea of what you want to do, it makes little difference whether you develop the interface or the underlying functionality first. Both can be readily modified and tested independently during development.

16.2 The Forms Development Subsystem

The Forms Development subsystem is an interface to the FMS forms editor and librarian utilities. It also provides form management functions that are not available through FMS. The forms you create and edit are stored in form libraries.

The main Forms Development menu, form FD, is a standard ALL-IN-1 menu that has options for creating, selecting, editing, reading, listing, and deleting forms from specified form libraries. As with other standard menus, Forms Development maintains current selections — in this case, a current form library and a current form in that library.

Form FD is illustrated in Figure 16-1, and the options are covered in the following paragraphs.

```
Jane Smith                                          Wed 24-Oct-1984
                          Forms Development

SEL     Select                        Library:
                                      Form:

C       Create
E       Edit
D       Delete
P       Print
R       Read
I       Index

CO      Copy a Known Form to Another Form Library
UP      Upgrade a V1.0 Form Library to V2.0 Format
EXT     Extract a Known Form and Create a New Form
RND     Reorganize Named Data
PCF     Precompile Form Library

Enter selection and press RETURN
```

Figure 16-1 Forms Development Menu

Select

The SEL option displays an overlay argument form that allows you to select a current form and a current library. The name of the library is validated here.

Most of the other FD options implicitly operate on the current form and the current library.

Create

The C option allows you to create:

- A new form
- A new data file from an entry form

  OR

- A DATATRIEVE record definition based on an entry form

These options, which are presented on overlay menu FDCREA when you select C, are described as follows.

FO – Create a Form

The FO option solicits the name of a new form and a library to store it in and then puts you in the FMS forms editor. You can optionally identify a standard form type to serve as a template. If you do specify a standard form, the template is extracted from form library OA$LIB:STANDARD.FLB, and the form you create with it is replaced in the library you have specified.

Regardless of how you start out, you eventually end up at the FMS Forms Editor menu. The options listed here, the FMS editing commands, and other FMS editing features are documented fully in the VAX FMS documentation set.

If you exit the FMS editor in the usual manner — that is, if you confirm the prompt that asks whether you want to save the form — the newly created form is added to the specified form library.

Although the FMS forms need not be stored in form libraries, ALL-IN-1 only recognizes forms in libraries and not individual forms.

FI – Create File from Form

The FI option is valid when the current form is an entry form with a valid .FILE specification. Entering FI calls the CREATE function with the current form name as a parameter. A new file is made to correspond to the current form. If the current form already has a file, using the FI option creates a new file that is empty.

Once the file has been created, the form and the file are collectively known as a data set. You can call up the entry form to add, change, and delete

records in the file, and you can use the information in the file in your application with data set references.

DD – Create DATATRIEVE Definition

Like FI, option DD is valid when the current form is an entry form containing a .FILE entry. This option creates a DATATRIEVE record definition based on the field attributes of the current form.

Entering DD first displays an argument form that requests the names of a DATATRIEVE dictionary and domain, and then it runs a procedure that builds the record definition based on the field layout and attributes of the form.

DATATRIEVE dictionaries, domains, and records are covered in detail in the VAX DATATRIEVE documentation set.

Edit

The E option allows you to edit:

- The screen image and attributes of the current form using the FMS forms editor
- A text description of the current form using your default text editor

  OR

- The Named Data portion of the form using your default text editor

These options, which are presented on the overlay menu FDEDIT when you select E, are described as follows.

S – Screen Image

The S option places you in the FMS editor with the current form. The only difference between this option and the Create Form option is that you are placed in the forms editor with an existing form instead of a new one.

You can change all aspects of a form with the forms editor, but it is best suited to modifying the screen image and the field attributes on a terminal that supports forms display. Small to moderate size lists of Named Data can be modified effectively using the S option, but it is more efficient to modify extensive lists with the T or ND options.

T – Text Description

The T option extracts an FMS Forms Description Language file from the current form and then places you in your default text editor with that description. This form description is an ASCII text description that corresponds to the binary version that serves as input to the screen-oriented forms editor.

If you exit the editing session in the usual manner, Forms Development calls FMS to translate the form description into binary form and update the form in the library.

Use the T option when you need to modify a form while running ALL-IN-1 on a terminal that does not support forms display. This option is also useful for modifying extensive (over 30 items) Named Data lists.

ND – Named Data

Like the T option, the ND option extracts an ASCII text description, but only of the current form's Named Data. The description is *NOT* a Forms Description Language file; instead it is an ALL-IN-1 feature available only through the Forms Development subsystem.

The Forms Description Language rules that you must follow while using the T option require that each Name/Data pair be in sequential, numbered order and not exceed 80 characters. The ND option allows you to edit Name/Data pairs as individual, unnumbered entries, with no length restrictions on the Data entry.

If you surround each separate Name element with double semicolons, all the text that follows up to the next set of double semicolons is interpreted as the corresponding Data element. For example:

```
;;.TYPE;;
MENU/CHOICE = CHOICE/CLEAR
/PRE = 'GET #LINE1 = "First Line" \GET #LINE2 = "Second Line"
\GET #LINE3 = "Third Line \GET #LINE4 = "Fourth Line"

;;CHOICE;;
.
.
.
```

In this example, .TYPE is treated as a Name element, and the three lines that follow it are treated as the corresponding Data element. Since this Data element exceeds the maximum allowable length for a single Named Data entry, the ND routine automatically creates two Named Data entries for .TYPE.

If you exit the Named Data edit session in the usual way, Forms Development translates the file into a standard Forms Description Language Named Data section and inserts it into the FMS Form Description Files. Forms Development then calls FMS to translate the description back into binary format and to update the form.

Delete

The D option deletes the current form from the current form library.

Print

The P option prints:

- A text description of the current form
- A listing of the current library

  OR

- The Named Data of the current form

These options, which are presented on the overlay menu FDPRIN when you select P, all call the QUEUE_PRINT function.

DF – Form Description

The DF option prints an image of the screen layout and a text description of the current form. The description is the same one presented by the Edit Text Description option.

You may specify an output device and the number of copies to print.

Use this option when you need an easily referenced version of all the attributes of a form.

LL – Library Listing

The LL option prints a list of all the forms in the current library. You may specify an output device and the number of copies to print.

Use this option to examine the size and contents of a form library.

ND – Named Data

The ND option prints a text description of the Named Data of the current form. The description is the same one presented by the Edit Named Data option.

Use this option when you need an easily referenced version of a form's Named Data.

Read

The R option displays:

- A text description of the current form
- A listing of the current library

  OR

- The Named Data of the current form

These options are presented on the overlay menu FDDISP when you select R. The FDDISP options are identical to those of the Print overlay, FDPRIN. The only difference is that the Read options all call the LIST function instead of QUEUE_PRINT.

Index

The I option lists the forms in the current library. The same function is performed by the LL option on the Read overlay menu.

Copy a Known Form to Another Form Library

The CO option moves an existing form to another library. This option displays an argument form that requests the name of the form library where the new form is to be placed, the name of the form to copy, and the name of the form library where the form to be copied is found. The command procedure verifies this information and then places a copy of the form in the new library. The known form is copied to the new library with the same form name.

You can use this option when you are customizing an existing form. By moving the form to another library (usually your USER.FLB or some other

personal form library), you can modify a version of the form while others continue to use the original form.

Upgrade a V1.X Form Library to ALL-IN-1 V2.X Format

The UP option upgrades all forms in an FMS Version 1.X form library to the current version of FMS. This option invokes an FMS conversion utility.

NOTE: ALL-IN-1 Version 1.X forms were created using FMS Version 1.X and must be converted to be compatible with ALL-IN-1 Version 2.X.

Extract a Known Form and Create a New Form

The EXT option allows you to create a new form from a known form stored in a library. This option allows ready duplication and adaptation of existing forms. This option is similar to the Copy option, but with EXT, you are placed in the forms editor with the extracted form. You must rename the form manually while in the forms editor if you wish to save it under a different name.

Reorganize Named Data

The RND option runs a procedure that sorts the current form's Named Data by the Name entry. Although the order of Named Data entries is functionally unimportant, Named Data becomes increasingly difficult to maintain as the size of it grows since entries usually are appended to the end of the list. Option RND provides a simple way to reorganize Named Data elements.

Precompile Form Library

The PCF option lets you create a compiled form library from an existing form library. Compiling a form library before using it improves the performance of ALL-IN-1. Use the Select option to choose the form library you want to compile.

16.2.1 Using Form Libraries

Where a new form is stored depends on how it is to be used. The common form libraries OA$LIB:MEMRES.FLB and OA$LIB:OAFORM.FLB are shared by all users. Thus any forms updated or created in those libraries may have an impact on all ALL-IN-1 users.

Personal form libraries are searched before OAFORM if they are specified in field FRMLIB of form PROFIL. Thus, you or any user can maintain any number of customized versions of OAFORM forms. If the forms have the same name, and the personal form library is searched before OAFORM, the customized version of the form displays when the form is called. ALL-IN-1 always searches OAFORM last unless you specify otherwise by entering "OA$LIB:OAFORM" in the FRMLIB field of the User Profile.

The System Manager's Guide also covers form library searches.

16.2.1.1 Updating Memory-Resident Forms in MEMRES.FLB.

OA$LIB:MEMRES.FLB contains frequently used forms that are linked into the ALL-IN-1 image when the image is built. Making a form memory resident allows it to be retrieved and displayed more quickly than if it were in a regular form library.

You can modify a memory-resident form like any other form, but you must relink the image to display the modified version. You can open MEMRES.FLB as a regular form library, but this serves no purpose since its contents are already memory-resident, and memory-resident forms are searched first when ALL-IN-1 looks for a match to the form name.

The following steps are required to modify a memory-resident form:

1. Use the Forms Development Edit option to change the layout, Named Data, or field attributes as desired and to update OA$LIB:MEMRES.FLB.
2. Create an object module from MEMRES using the following FMS command at DCL level:

   ```
   $ FMS/OBJ/OUT=OA$BUILD:MEMRES OA$LIB:MEMRES.FLB
   ```

   where the /OUT qualifier identifies the ALL-IN-1 sources directory.
3. Relink the image. This must be done outside of ALL-IN-1:

   ```
   $ @OA$BUILD:A1LNKDRV
   ```
4. Copy the new image to OA$LIB using the command:

   ```
   COPY A1.EXE OA$LIB:A1.EXE
   ```
5. Install the new image as described in the *ALL-IN-1 Installation Guide.*

The modified form will be memory resident when the image has been created. You can add forms to MEMRES, but you should only do this when the form is used often.

The following steps are optional. They increase the performance of ALL-IN-1 by performing most of the parsing and organizing of a form definition. This is also referred to as precompiling.

6 Run the ALL-IN-1 image.

7 At a menu choice field type:

```
<OA$FBT_WRITE form-library-name
```

where form-library-name is the name of the form library you want to precompile.

This function creates a copy of the form library with a .FLC extension. When ALL-IN-1 is initialized, if the file with the .FLC extension does not match the .FLB file, an error appears on line 24 of the screen.

Instead of using the OA$FBT_WRITE function, you can use the Select option on the Forms Development menu to select a form library. Then, use the PCF option to precompile that form library.

8 Use the OA$FBT_REMOVE function to delete previously installed forms from the globally shared section.

9 Use the OA$FBT_INSTALL_LIBRARY function to install new libraries in the globally shared section. Installing form libraries as globally shared sections between users increases performance. Memory is saved when only one copy of a form library is needed for all users.

You can perform the previous, optional steps on any form library, not just OA$LIB:MEMRES.

16.2.2 Testing Newly Created Forms

After you create a form, you can test it immediately. If you replace the form in OAFORM or in a library specified in User Profile field FRMLIB, the NEWLIB function call that Forms Development makes automatically opens the newest version of the library. If you replace the form in a library that has not yet been opened by ALL-IN-1, make the following function calls interactively:

```
<DUMP_CACHE \OA$FLO_OPEN_LIBRARY library_name
```

DUMP_CACHE clears out all old versions of the form that ALL-IN-1 may have been maintaining in cache, and OA$FLO_OPEN_LIBRARY opens the specified form library.

You can then enter the form name in a choice field, and it displays. If any errors occur during Named Data parsing, the message:

```
Error occurred while form 'formname' was current
```

appears. Pressing Gold W then displays any related messages and probably pinpoints the cause of the error.

16.3 The Programming Functions Subsystem

The Programming Functions subsystem provides options for creating, and editing program modules. Modules are stored as File-Cabinet documents, and the Format corresponds to the language in which the the module is written.

The main Programming Functions menu, form PF, is a standard ALL-IN-1 menu that is similar to the Word and Document Processing menu. As with Word and Document Processing, Programming Functions maintains a current selection (a module instead of a document) and a folder where it is stored.

Form PF is illustrated in Figure 16-2, and the following paragraphs cover the options.

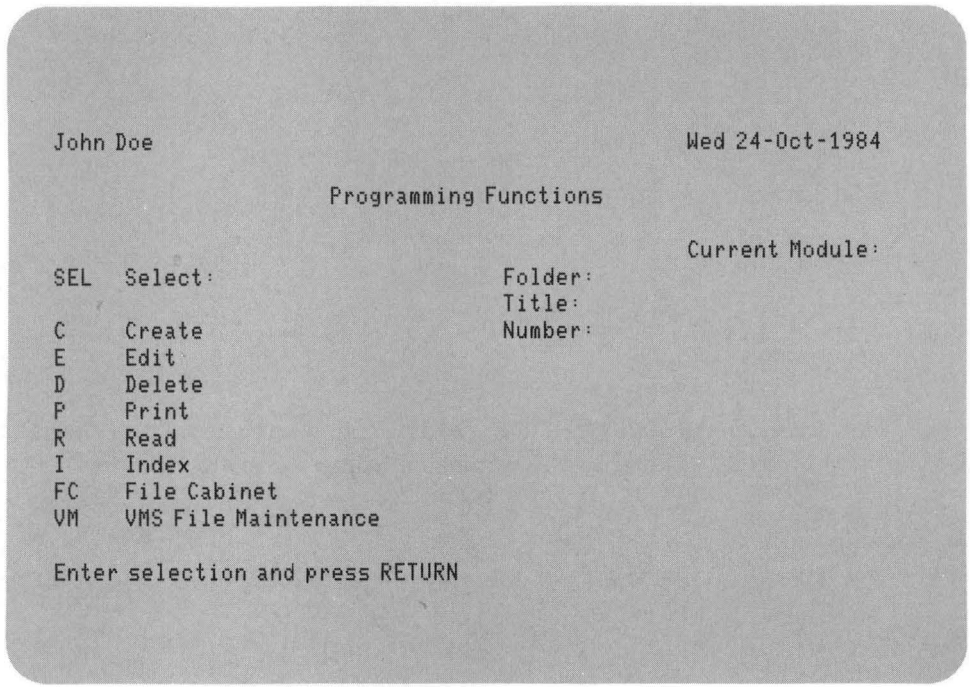


Figure 16-2 Programming Functions Menu

Select

The SEL option allows you to select a current module or program to process. Most other options implicitly work on the current module.

Create

The C option allows you to create a new module. The module is created and stored as a File-Cabinet document, and the format you give it corresponds to the language it is written in.

When you create a module, ALL-IN-1 calls the EDIT function for the initial creation/edit session.

Edit

The E option allows you to edit the current module. ALL-IN-1 calls the EDIT function with your default edit style.

Delete

The D option deletes the current module.

Print

The P option calls the QUEUE_PRINT function to print the current module. You can specify the number of copies to print.

Read

The R option calls the LIST function to display the current module at the terminal.

Index

The I option displays a list of modules. The list is based on search criteria that you specify.

File Cabinet

The FC option displays the File Cabinet menu, which presents a list of options for maintaining program modules within the File Cabinet.

VMS File Maintenance

The VM option displays form FILEM, which provides options for manipulating and maintaining RMS files without leaving ALL-IN-1.

16.4 Other Development Tools

Several features of ALL-IN-1 simplify applications development. These features are covered in the following sections.

16.4.1 Debugging Features

ALL-IN-1 provides several debugging tools which are useful in error tracking or when you need to break a procedure down, and watch the processing step by step, as it occurs.

- The COMMAND function is one of these debugging tools. If you call the COMMAND function interactively to execute DCL command procedures, you can observe the results before you associate those command procedures with menu options or other form-based applications. OA$LIB:V.COM is a debugging tool to use with the COMMAND function. It is a short file that runs other command procedures in DCL VERIFY Mode. To use it, specify:

  <COMMAND V com_file

  It runs com_file in VERIFY Mode. VERIFY Mode echoes command procedure lines as they execute and performs most symbol and lexical-function substitution before displaying a line. When the main process assigns values to global DCL symbols through WRITEs to DCLMAILBOX, those values also display.

- Another debugging function is OA$MERGE_LINE. It allows you to test template commands directly, without actually creating a test document. See Chapter 6 for a description of this function.

- ALL-IN-1's Logging facility, which is described in the *System Manager's Guide*, is also useful in debugging. Another ALL-IN-1 facility, Tracing, provides a way to monitor different functions during execution. Both the OA$TRA_SET function and the Trace menu accessed by GOLD % invoke the Tracing facility.

- Another debugging tool is Gold (, which issues the DCL SET VERIFY command in the subprocess. .TRACE_ON, a Script-facility feature, is also activated by pressing Gold (. .TRACE_ON echoes ALL-IN-1 script lines as they execute.

16.4.2 Using the Subprocess Interactively

The DCL function places you at DCL level in the subprocess. You can call this function from any field in any form by pressing Gold $. You can also call DCL from a menu by entering a $ or <DCL.

Use this function interactively to:

- Run VMS utilities and other layered products.
- Test personal programs.
- Browse through directories and logical-name tables.
- Move or rename files.
- Set privileges and/or file protection.
- Show process and system information.
- Use VMS help.
- Execute other VMS or DCL commands.

See the VAX/VMS documentation set for a full listing of DCL commands.

16.4.3 Using the Editor

The DCL function is particularly useful when called from within the default editor. Pressing Gold $ or entering DCL at the Line or Command Mode prompt places you in the subprocess at the $ prompt. If you are currently editing a command procedure, you can test DCL commands interactively before adding them to the file. You can also create listings using /OUT qualifiers on DCL commands and include those listings in the current file.

If you are developing a program and need to use a calculator, the desk calculator is readily accessible by pressing Gold #. The final result, if any, that appears in the display register is placed in the primary paste buffer and can be recalled by pressing the PASTE key.

You can test other components of an application while editing by pressing Gold M to call the Editor menu or Gold I to call the Interrupt menu. If you are editing a text form description, for example, you can test the form in ALL-IN-1 without leaving the editor:

1 Create a temporary file from the current contents of the main edit buffer while in Line (CTRL/Z) or Command (Gold keypad 7) Mode:

```
Command: write TEMP.FLG
```

2 Enter interactive DCL to translate that form description into binary form for inclusion in a forms library:

```
Command: DCL{ENTER}
$ FMS/TRANS TEMP.FLG
${CR}
```

File TEMP.FRM now exists in your default directory, and it contains a binary version of the form you are currently editing. (The name of the file need not be the same as the name of the form.)

3 Press Gold M (from Change Mode) to call the Editor menu. Now you can call any ALL-IN-1 function interactively. Assuming you want to replace the form in a library that is currently open (for example, USER.FLB), close that library manually and replace the form:

```
<OA$FLO_CLOSE USER \DCL FMS/LIBRARY USER TEMP
```

These two function calls close OAUSER:USER.FLB and update (or add) the form in TEMP.FRM to that library.

4 Call the DUMP_CACHE function as a precaution and reopen the form library:

```
<DUMP \OA$FLO_OPEN USER
```

The form can now be displayed as an ALL-IN-1 form. Enter its name in the choice field of the Editor menu to test it.

5 Press EXIT SCREEN at the Editor menu when you wish to return to the editing session, (either to file the form description or to continue to modify it).

16.4.4 Using the GET Function

Interactive GET function calls allow you to:

- Bypass the entry-form interface and retrieve any field or series of fields on any form.
- Examine the current value of any symbol, allowing you to compare that value to information that may be displayed on the current form.
- Load any symbol-table symbol or field on the current form with a value, allowing you to set up a test environment for an application.

16.4.5 Using View Forms

The view forms are overlays that provide a window to the underpinnings of an application.

OA$VIEW_FIELDS, which is displayed by pressing Gold V, presents a table of the names and locations of every field on the form. Use it while on an entry form, for example, to disclose more information about the layout of the fields on the form (and thus the fields in the associated data file).

OA$VIEW_ND, which is displayed by pressing Gold N, presents the Named Data of the current form or form set. Use it to examine the function calls and field qualifiers that are invoked when you call a form or make an entry in a field.

OA$VIEW_MESSAGES, which is displayed by pressing Gold W, displays all messages that were generated in conjunction with the last error or informational message. Use it to get more information about an error message.

For example, when the following message displays: `Error occurred while form 'formname' was current`, press Gold W to see why the error occurred. Often an underlying parsing error message identifies the exact location of the error in Named Data.

16.5 Application Design Guidelines

An application can be implemented using any combination of ALL-IN-1 and VMS functionality. Most applications take one of the following general formats:

- A standard application that uses standard ALL-IN-1 forms as the interface and implements most of its functionality through function calls and scripts.
- A program that runs in the subprocess and that is accessed through a single menu option, COMMAND function call, or DCL function call. The program handles all user interface, usually through WRITEs to OAMAILBOX that display standard ALL-IN-1 forms. It may also call other functions through OAMAILBOX, but the bulk of the functionality is provided within the program.

- A user-defined function that has been integrated into the ALL-IN-1 image. The application is implemented as a function call.

These three types of applications are covered in the following sections.

16.5.1 Standard Applications

A standard application is generally built as a menu tree. The top-level, or subsystem menu is a standard ALL-IN-1 menu. Subsystem menu options may make FORM function calls to display other forms, possibly other menus; or they may call some other function such as COMMAND or DO to perform a task. Additional options display on overlay menus.

The Forms Development subsystem is an example of a standard application. It is accessed by a standard menu, FD. Several options (C, E, P, R) display overlay menus, others (SEL, CO, UP, EXT, RND) display nonmenus, and the rest (D, I), do not display forms at all. All options eventually perform some task or group of tasks.

When you give the top-level subsystem menu the same name as the menu option that calls it, the location of that menu option becomes a minor consideration. You can position the application anywhere in ALL-IN-1 since the Form Processing facility eventually treats any interactive user input as a form name if it is not meaningful in some other manner.

Also, you usually should not redefine keypad 0, the EXIT SCREEN key. The consistency of this feature is important: this key will get the user out of anything.

Most tasks that are not performed directly from the Named Data of a menu can be performed directly from the Named Data of the various nonmenu form types. Use entry forms to maintain data files, use search and select forms to search those files, and use argument forms to obtain parameters and symbol values.

A Special Features and Restrictions section accompanies most ALL-IN-1 function descriptions in Chapter 6. These sections provide guidelines on how to use functions in standard applications.

16.5.2 Standalone Applications

When you develop a standalone application that runs in the subprocess, you may call standard ALL-IN-1 forms through OAMAILBOX to serve as the user interface.

If you call the FMS form-driver directly in your application, any Named Data entries that are unique to ALL-IN-1 are ignored. Also, many form-related features would not be supported, including form types, form and field qualifiers, and keyboard definitions.

NOTE: ALL-IN-1 does not use FMS function-key User Action Routines (UARs). Instead, it leaves this facility available to user applications. It is important to note that forms converted from FMS Version 1.0 format to Version 2.0 format insert a reference to a function-key UAR that emulates Version 1.X keypad keys. This reference should be removed to allow full utilization of the keyboard in Version 2.X ALL-IN-1 applications.

Most application programs that run from a VMS command procedure also run under the Office Menu in the subprocess. If your program does not depend on ALL-IN-1 for any of its functionality, you can simply add an option to an existing menu that runs your application. You can also communicate with ALL-IN-1 through mailbox WRITEs and READs and can thus call ALL-IN-1 functions.

16.5.2.1 Communicating through Mailboxes. Applications that are not linked into the main process may communicate to Office Menu through mailboxes from the subprocess. Mailbox WRITEs and READs are used primarily to call Office Menu functions from the application and return any results to the caller.

The following example shows how to implement a WRITE and READ, from the subprocess, in BASIC.

BASIC

```
100 OPEN "OAMAILBOX" FOR OUTPUT AS FILE 1%
200 OPEN "DCLMAILBOX" FOR INPUT AS FILE 2%, ACCESS READ
300 PRINT #1%, "OA GET OA$TERMINAL"
308 ON ERROR GOTO 900
310 INPUT #2%, A$
311 PRINT "Response = " ; A$
314 GOTO 310
900 RESUME 910
910 CLOSE 1% , 2%
999 END
```

NOTE: *You must open DCLMAILBOX with Read Only access, no matter what language you are using.*

16.5.3 User-Defined Functions

When an application is significantly useful to other applications, it should be integrated into ALL-IN-1. A synergy is formed when an application is implemented as a user-defined function: all existing applications benefit when a new application provides new, shareable functionality.

Two existing applications that are integrated into the system at this level are the Communications Control application (the CXP function) and the Time Management subsystem (the CALENDAR function).

The following subsections present guidelines to follow when you implement an application as a user-defined function.

16.5.3.1 VAX/VMS Requirements. If you need to write a program or programs, select a language which can run standalone or be called as a subroutine. The generated code from the language should also be shareable.

Follow the standards for developing VAX applications as outlined in the VAX/VMS documentation set. Experiences gained in moving ALL-IN-1 from V2.0 of VMS to V3.0 and from V3.0 to V4.0 suggest that compatability is generally maintained if the documented guidelines are followed.

Even with interactive programs, you should not assume that the program will be talking directly to a terminal. Within ALL-IN-1 there are times when the program is talking to a mailbox instead of to a terminal. Therefore, in dealing with a terminal, use the logicals SYS$INPUT and SYS$OUTPUT.

16.5.3.2 Modularity. Develop your application to be as modular as possible. That is, the application should first be analyzed to identify the individual functions, and these functions should then be implemented as separate callable routines.

The advantage of modularity is that the application can be more easily customized to suit a particular set of needs. Also, as user needs change, the modules can be combined in different fashions.

It is usually most effective to develop modules as subfunctions. That is, implement discrete tasks as separate callable routines. Each routine should correspond to a subfunction of the main user-defined function.

16.5.3.3 Calling Standards for User-Defined Functions. If you design the application to run standalone, it should have a callable interface so that it can be directly linked to ALL-IN-1. The individual function modules should also have global entry points so that ALL-IN-1 can control the flow.

There should be a single source file for both the standalone and integrated versions. Where possible, choose a language that has conditional assembly capabilities. Otherwise, a simple filter program can be developed that does a split of the main source file into two separate files which can be independently compiled: one for standalone, one for callable. Ideally, through proper design, there should be no need for conditional assembly.

In the callable version of the program, keep the following in mind:

- Do not use the VMS EXIT System Service. Instead, do a normal return from the main routine.
- Do not assume that you can leave files open, AST's active, EXIT handlers active, or virtual memory in use. If the application is callable, it is likely to be called more than once.
- Use the VMS message facility.
- Use standard System Service and Run-Time Library calls where possible. There are many cases where one facility calls another facility to perform a specific subfunction. Thus, an AST that may be valid during some parts of an application may cause unpredictable results while in another. If AST's are required, be sure that they are not active when code that is not specifically designed to react to the AST is active.

16.6 Performance Considerations

Keep the following performance considerations in mind while developing an application.

Command Procedures Versus Scripts

When you can perform a task through either a command procedure or a script, use a script. You will avoid the overhead associated with WRITEs and READS from the subprocess.

Also, if you have one or two lines that must be executed at DCL level, use the OA$DCL special GET destination instead of writing a short command procedure.

Special Symbols

When file-based information is also stored in a special symbol, retrieve the value with a special symbol instead of with a data set reference. For example, ALL-IN-1 stores the job title of the current user in User Profile field TITLE and also in special symbol OA$TITLE. You can fetch the name faster by specifying "<GET OA$TITLE" than by specifying "<GET PROFIL.TITLE[OA$USER]".

FMS Field Highlights

The simpler a form, the faster it is processed. Forms with extensive video attributes and field highlights take longer to display than forms that have the default FMS attributes.

Also, the /HIGHLIGHT field qualifier should be used sparingly if performance is a consideration.

RMS Tuning

When creating complex data files, use RMS file-tuning techniques to improve performance during entry-form-based file updates. RMS file tuning is covered in detail in the VAX/VMS documentation set.

Other performance considerations are covered in the *System Manager's Guide.*

16.7 Documenting a Customized System

16.7.1 User Documentation

The user documentation for an application should let the first-time user know what is required to run the application, what options are available, and a description of what action the user should take on error messages. See the *Style Guide* and the *Writer's Guide* for information about writing your own documentation.

16.7.2 Help Documentation

If you have customized your ALL-IN-1 system, you may also want to customize HELP for your new applications. Following are some guidelines that cover style considerations and the kind of information to provide. Some examples of HELP templates which illustrate the current ALL-IN-1 HELP style are provided. Examples of DIGITAL Standard Runoff HELP Templates are also provided.

16.7.2.1 Style. The primary style considerations in writing HELP are space and audience. Only eight lines of text display on each HELP screen at a time, and the fewer HELP screens necessary to convey a HELP message, the better. Since HELP messages must often convey a lot of information, they must be concise.

At the same time, however, those who use HELP most often are users new to computers or new to ALL-IN-1. HELP users are not necessarily familiar with computer terminology, nor with ALL-IN-1 terms like "menu", "option", or "create." Substitute real-life terms whenever possible, and when computer terms are necessary, try to make them self-explanatory.

For example, ALL-IN-1's Create option in the Word and Document Processing is easy to understand: the user uses that option to create a new document. But in the interest of maintaining consistency between menus and procedures, ALL-IN-1 also uses the Create option on the Meetings Menu. The user uses this Create option to *schedule* a meeting; thus it is important to make that connection in the HELP message.

Sometimes you can use the titles of the HELP messages to help explain the procedure, instead of simply naming the option or field described in the message. For instance, instead of titling a HELP message, "The Read Option," name it "Reading a Message," or instead of "The Title Field," call it "Entering the Document Title." Task-oriented titles tell the user what to expect, and appear in the Additional Topics as a list of tasks the user can recognize.

Making HELP message text "user-friendly" as well as informative and concise can be difficult. For example, a typical "user friendly" HELP message might read:

```
If you want to Read the message you have received, use the Read option on the
Electronic Messaging Menu. You must enter R and press RETURN to read your
message.
```

But by using imperatives and eliminating the use of the second person for example, the message can still be concise without sacrificing "friendliness." Plus, it is usually not necessary to tell the user what menu or form he or she is using, except of course in the level- 1 HELP that introduces the menu or form to the user.

Thus, the following message conveys all the information the user needs to know, and is still "user friendly" in that it is straight forward and non-complicated:

```
Enter Read (R) to read the message.
```

16.7.2.2 Context. It is important to have a clear understanding of what it is that the user wants to know when he or she presses the HELP keys. In general, when a user requests HELP:

- At the menu level, they want to know WHY they should choose an option
- At an entry form level, they want to know HOW to complete an entry.
- At a prompt, they want to know what to do next
- At an error message, they want to know how to correct the error

Sometimes it is necessary to provide both the WHY and the HOW, and sometimes it is necessary to provide even more information. It is a good idea to go through the procedure you are writing about in order to understand what information the user needs.

For example, if the user enters the Index (I) option from the Word and Document Processing Menu and presses RETURN, ALL-IN-1 displays the Document Index form. The cursor is positioned in the Folder field at the bottom of the form. Normally, a request for HELP on an entry form at this point would give the user a message like:

```
Enter the Folder name in the Folder field and press RETURN.
```

However, in this case, more information is required, because the index procedure requires more. ALL-IN-1 does not automatically provide an index: the user must first complete the fields providing the criteria

ALL-IN-1 uses to compile the index, and the HELP message should convey that information. The actual HELP message reads:

```
Listing Documents by Folder

To complete the Index form, enter the fields you want to include in the
search and press RETURN, or just press RETURN to search the folder currently
displayed.

To see a list of documents in the current folder, press RETURN.
```

16.7.2.3 Templates. Following are templates that illustrate the current ALL-IN-1 HELP style. Use them as guidelines to writing HELP. Given the complexity of ALL-IN-1 and the range of its features, it is not always possible to follow the templates exactly, but the more consistent the format, the easier it is to update and customize HELP.

HELP on menus (level-1):

```
1 WP
Word and Document Processing

The Word and Document Processing menu lists a set of options for producing
and manipulating documents stored in your File Cabinet.

Use the Word and Document Processing options to work with messages,
letters, reports, and other documents.

...
```

HELP on menu options (level-2):

```
2 SEL
Selecting a Document

Enter Select SEL to make a document the current document. Complete the
Selecting Current Document form.

To work on a document in your File Cabinet, it must be the Current Document.
The Current Document Block on the Word and Document Processing menu
displays the header information of the current document.

...
```

HELP on general information topics (level-3):

```
3
Selecting a Document Using the Function Keys

You can select documents from your File Cabinet quickly and easily by using
the File Cabinet function keys. You can use these keys whenever you are on a
subsystem menu.

The File Cabinet function keys are:

. . .
```

HELP on an entry form (level-1):

```
1 DOCDSC
Creating a Document

Use this form to enter the title and the folder name of the document you want
to create. You can also enter keywords to identify the document. Press TAB
to move forward through the fields, and BACKSPACE to move back. When all of
the information is correct, press RETURN.

. . .
```

HELP on an entry form field (level-2):

```
2 FOLDER
Entering the Folder Name

Enter the name of the folder in which you want to file the document.
ALL-IN-1 usually supplies a folder name in this field. Change it, or create
a new folder by entering a new folder name. Press TAB to . . . .
```

HELP on a system prompt (level-1):

```
1 PRMPT_CREATE
Document does not exist

Check the document and folder name you entered for accuracy. Or, answer Y to
create a new file.
```

HELP on an error message (level-2):

```
2 WPL_ABORT
Fatal error, edit session aborted

You cannot continue this editing session. Go back to the Word and Document
Processing menu and begin again, or see your system manager for further
information.
```

16.7.2.4 DIGITAL Standard Runoff HELP Templates. Use the following ALL-IN-1 HELP templates to add new HELP modules to the HELP library. RUNOFF commands and cross-referencing commands are already in place. You insert the Keyword, title, text, and cross-reference list in the areas marked with brackets.

Note that every level-1 module lists a set of format RUNOFF commands. These must be included in every level-1 HELP module.

```
.page size,75
.no paging
.autoparagraph
.fill
.justify
.set paragraph 0,1
.break
1 [Keyword]
.break
[Title]
.paragraph
[Text
.
.
.]
.keep
.break; = [Cross-Reference Keyword - optional]
.break; = [Cross-Reference Keyword - optional]
.nokeep
.fill
```

Figure 16-3 Level-1 HELP Module Template

```
.break
2 [Keyword]
.break
[title]
.paragraph
[Text
.
.
.]
.keep
.break; = [Cross-Reference Keyword - optional]
.break; = [Cross-Reference Keyword - optional]
.nokeep
.fill
```

Figure 16-4 Level-2 HELP Module Template

```
.break
3 [Keyword]
.break
[Title]
.paragraph
[Text
.
.
.]
.keep
.break; = [Cross-Reference Keyword - optional]
.break; = [Cross-Reference Keyword - optional]
.nokeep
.fill
```

Figure 16-5 Level-3 HELP Module Template

A

Exercises

A.1 Introduction

This appendix provides exercises for common ALL-IN-1 tasks. The exercises range from user-oriented activities to programming-level activities. Several of them are examples of application development.

The following is a list of exercises:

- Using standard ALL-IN-1 subsystems
- Modifying the User Profile (user level)
- Advanced user tasks
- Modifying the User Profile (programmer level)
- Accessing the subprocess
- Using functions interactively
- Using the Hard Copy interface
- Using development tools

- Modifying a Menu Form
- Creating an Entry form for an indexed file
- Creating a Select Form
- Modifying a script
- Creating a shared template
- Using DATATRIEVE

A.2 Exercise 1 — Using Standard ALL-IN-1 Subsystems

A.2.1 Objective

To gain a basic understanding of the most common functions provided by ALL-IN-1 from the user's perspective.

A.2.2 Preparation

Log in to VMS and run ALL-IN-1. Your system manager can provide you with the specific account and password information.

A.2.3 Exercise

Perform the following tasks. Use the Help key and follow the menus as a guide.

1. Create and send a mail message to one or more users.
2. Read a message.
3. Create a document under Word and Document Processing.
4. Edit the document just created.
5. Print the document.
6. Use the Word and Document Processing Index option.
7. Use the Desk Calculator.

A.2.4 Summary

At this point, you should understand the basics of ALL-IN-1.

A.2.5 References

ALL-IN-1 Office Menu User's Reference

A.3 Exercise 2 — Modifying the User Profile

A.3.1 Objective

To gain an understanding of what the User Profile does in ALL-IN-1. This exercise updates only the portion of the User Profile that is accessible to all users.

A.3.2 Preparation

You should be logged in to VMS and ALL-IN-1. From any menu, enter US to display the User Setup Parameters menu. Select the UP option to display form PROFILE.

A.3.3 Exercise

Change the following information in your User Profile record. Press TAB between fields; press RETURN when you finish making your changes.

1. Full name
2. Department
3. Title
4. Calendar work week

Note that some of your changes are evident at the top of the starting form.

A.3.4 Summary

By modifying your User Profile record through form PROFILE, you have changed your personal ALL-IN-1 environment the way most users would. More extensive changes can be made by modifying the Profile through form-set PROFIL, as discussed in Exercise 4.

A.3.5 References

ALL-IN-1 Office Menu User's Reference

A.4 Exercise 3 — Advanced User Tasks

A.4.1 Objective

To gain a more thorough understanding of ALL-IN-1 and to introduce short cuts for the advanced user.

A.4.2 Preparation

You must be logged in to VMS and running ALL-IN-1.

A.4.3 Exercise

Perform the following tasks. Use Help and the *User's Reference* as needed.

1 Recognition — Go to Word and Document Processing, choose the Select (SEL) option. In the title field on the Select overlay, press Gold L.

2 Direct access to forms — Type the name of a form to go directly to that form. Try the following:

- WP — Word and Document Processing menu
- FC1 — File Cabinet Options menu
- WPINDX — Word and Document Processing Select form
- PROFILE — User Profile Entry form

3 Additional options — Press Gold A at the Main Menu and several subsystem menus to see what additional option menus are available. Use Gold V to determine the name of the additional option forms and call those forms directly. Use the EXIT SCREEN key to recall the previous form.

4 Stacking Options — Enter a sequence of menu options from one or more forms. Include some that call user entry forms.

5 Formats and Handling — Create a document using a special handling or format.

6 Selection Lists — Use the WP Index option to automatically create a selection list. Next use the selection list with the WP Select option. (Keypad "." is the NEXT LIST key.)

7 Mini-Calculator — Press Gold # in any field in any form, and do some calculations.

8 File Cabinet Options — Enter FC at the WP menu. Refile the current document in another folder named TEST1, and cross-file that refiled document in a folder named TEST2.

9 Calendar Options — Enter Time Management. Create appointments for today and for one week from today, and schedule a meeting with two or more people.

A.4.4 Summary

You should now have a good working knowledge of most of the functions an office professional is likely to perform.

A.4.5 References

ALL-IN-1 Office Menu User's Reference

A.5 Exercise 4 — Modifying the User Profile (Programmer Level)

A.5.1 Objective

To develop an indepth understanding of how the User Profile affects your ALL-IN-1 environment.

A.5.2 Preparation

Perform this task from the ALL-IN-1 system manager's account by running ALL-IN-1 with the user name MANAGER. Once your Profile record has been changed, you may run ALL-IN-1 again using your own user name.

A.5.3 Exercise

At the System Manager menu, select PROFILE as the current master. Then select the Edit option. Form set PROFIL appears. Note the similarity to form PROFILE: both forms are associated with data file OA$DATA:PROFILE.DAT, but only PROFIL describes the data file fully.

Enter your ALL-IN-1 user name in the key field and change the following fields:

1 Starting menu (Note: enter another menu name)

2 Editor style

3 Form libraries - Change to "OA$LIB:MANAGER,USER"

4 Privileges - Change the following to "Y":

   a DCL

   b CMD

   c SRC

5 Go to the last field on the first page of PROFIL, and press the TAB key again. The second page of the form set (PROFIL2) displays. Press Gold T to redisplay the first page, or press RETURN to update the User Profile.

A.5.4 Summary

You have now performed an ALL-IN-1 system management task. You have given yourself full ALL-IN-1 privileges and access to the system manager form library.

To test that you have updated your User Profile record correctly, run ALL-IN-1 with your name and enter $ in the choice field of the starting menu. Now enter DCL commands. Go back to the menu and enter SM. The System Manager menu should display.

A.5.5 References

ALL-IN-1 Office Menu System Manager's Guide

A.6 Exercise 5 — Accessing the Subprocess

A.6.1 Objective

To gain experience accessing the subprocess from ALL-IN-1.

A.6.2 Preparation

You must have the DCL privilege set in your User Profile. You cannot access the subprocess directly without this privilege.

A.6.3 Exercise

Type Gold $ from any point in ALL-IN-1. The subprocess $ prompt appears. You may now issue DCL commands and execute DCL command procedures. To return to ALL-IN-1, press RETURN at the $ prompt.

If you are on an ALL-IN-1 menu form, you may type a $ to use subprocess commands. Once again, press RETURN at the $ prompt to return to ALL-IN-1

The subprocess may also be accessed from a menu by typing DCL. The $ prompt appears. Press RETURN at the $ prompt to resume ALL-IN-1 processing.

Try executing DCL commands using each of these methods of Accessing the Subprocess.

A.6.4 Summary

You have now practiced accessing the sub-process from within ALL-IN-1.

The subprocess is a convenient method of executing VMS commands and command procedures without leaving ALL-IN-1.

A.6.5 References

ALL-IN-1 Office Menu Application Programmer's Reference

A.7 Exercise 6 — Using Functions Interactively

A.7.1 Objective

To develop expertise in using ALL-IN-1 functions directly.

A.7.2 Preparation

Using ALL-IN-1 functions requires a solid understanding of what occurs when they are invoked. To acquire this knowledge, refer to Chapter 6 of this guide. In addition, you must have the ALL-IN-1 CMD privilege to perform these exercises.

A.7.3 Exercise

NOTE: *You must have the ALL-IN-1 CMD privilege to perform these exercises.*

Perform the following:

1. Use the DISPLAY function to have ALL-IN-1 display text of your choice on line 24.
2. Use the LOGICAL function to show the setting of OA$LIB.
3. Use the LIST function to list a VMS file of your choice.

4 Use the COMMAND function to execute a DCL command procedure. (This exercise requires the DCL privilege in your ALL-IN-1 User Profile.)

5 Use the OA$FLD_EXIT operator to simulate the EXIT SCREEN key.

You can look for a form that is different from your current form. Try using the ALL-IN-1 HELP function. (Remember to use the HELP function, not the HELP key.)

A.7.4 Summary

Now you have had the opportunity to become acquainted with some common ALL-IN-1 functions. This exercise demonstrates the interactive capability of the functions. In addition, you may use them in script files.

A.7.5 References

ALL-IN-1 Office Menu Application Programmer's Reference

A.8 Exercise 7 — Using the Hard Copy Interface

A.8.1 Objective

This exercise is designed to familiarize you with the hard copy format of ALL-IN-1. Use Hard Copy Mode when the terminal does not have a video screen, or the video workstation does not support DIGITAL standard screen attributes.

A.8.2 Preparation

You should be experienced with using ALL-IN-1 in Screen Mode. Hard Copy Mode prompts are shorter than the standard menu format, so you will be better able to use Hard Copy mode if you are already familiar with the standard format.

You do not need a special terminal for Hard Copy Mode. You can use this interface on the workstation where you do your everyday work.

A.8.3 Exercise

Type <MODE C from ALL-IN-1's Main Menu. You will see the following prompt:

```
Form) Main Menu
    ) Options: WP,EM,SM,TM,IM,BA,COM,M,TR,EX)
```

You are still on the Main Menu, but in Hard Copy Mode. As an example of moving from menu to menu, type DIR to go to the Directories menu. Press EXIT SCREEN to return to the Main Menu. (If you have no EXIT SCREEN key, just type the word Main to return to the Main Menu.)

Now try these exercises:

Press the ESCAPE key twice. This gives you a snapshot of the current menu. While in Hard Copy Mode, you use the double ESCAPE to display the current form.

You simulate Gold-keys and keypad keys on a hard-copy terminal by using the ESCAPE key. To return to the previous menu, press ESCAPE and a 0. This simulates the EXIT SCREEN key that usually does not exist on a Hardcopy terminal. Any of the usual keypad keys are simulated this way. For example, press ESCAPE and H to use the Help facility, followed by ESCAPE 0 to return to your previous menu. Another sample can be tried using the scratch pad with ESCAPE.

Experiment with other forms and subsystems to see how Hard Copy Mode works. For example, try the Desk Calculator. Notice that it behaves quite differently from Screen Mode. In Hard copy the Desk Calculator only supports Formula Mode.

Read a document from the Word and Document Processing menu.

If you have DCL or CMD privileges in your User Profile, you can perform these tasks in Hard Copy Mode also.

If your terminal supports the full-screen interface, type <mode s to return to screen mode.

A.8.4 Summary

In general, ALL-IN-1 commands and menu options are performed identically in the Hard Copy and screen Interfaces. This exercise emphasizes the differences between the two.

A.8.5 References

ALL-IN-1 Office Menu Application Programmer's Reference

A.9 Exercise 8 — Using Development Tools

A.9.1 Objective

To become familiar with the tools available within ALL-IN-1 to develop and modify customizable applications.

A.9.2 Preparation

You should be able to use the FMS utility to do form processing. It is also necessary to know what form libraries are, and how to access forms in those libraries.

A.9.3 Exercise

Type GOLD % to display the Trace Argument form. Select some trace functions, and request the option: `Trace output to terminal`. Running ALL-IN-1 with trace activated, allows you to see how much processing occurs every time a user enters input.

Use the Forms Development menu to create or edit a form. Experiment with the available ways to edit forms.

Compare the output generated by the three different methods of printing available on the Forms Development menu.

If you work in a high-level language that is installed on your system, create or edit a source module or program from the Programming Functions menu. Modules can be treated as documents in the File Cabinet.

A.9.4 Summary

ALL-IN-1 provides many different application tools. Anyone who has to write or maintain ALL-IN-1 applications should be aware of the additional capabilities ALL-IN-1 provides. Chapter 16 is a good introduction to this information.

A.9.5 References

ALL-IN-1 Office Menu Application Programmer's Reference

VAX-11 FMS documentation set

A.10 Exercise 9 — Modifying a Menu

A.10.1 Objective

To make a simple change to a menu form and its associated Named Data.

A.10.2 Preparation

You should be familiar with the FMS screen editor. This exercise uses both the layout and Named Data portions of the editor.

You will also need system manager privileges, and you should have a user form library (USER.FLB). Using the Forms Development menu, copy form DIR from OAFORM to your user form library.

A.10.3 Exercises

Using the Forms Development menu, add an option named TY to the screen with the text Thank You.

Use the FMS editor's D option to edit the Named Data. Move to the first blank named data line. Put TY on the name line, and following on the data line: DISPLAY You are welcome.

Exit the FMS editor. Be sure to save the changes you have made. Type DIR on the Forms Development menu. You should see the newly modified directory menu. Use the Thank You option (TY) and notice the response.

To return to using the standard DIR menu, delete or rename the menu you have created from your USER forms library.

A.10.4 Summary

Although this is a simple example of changing a form, you have added a new option to a menu. Many of the more functional changes you make to a menu are just as easy. If you have a working knowledge of FMS, you can modify ALL-IN-1 menus.

You do not need to change the screen layout to modify the Named Data. For example, if you wish to modify your Thank You option to display a form or a different message, you need only change the Named Data. Because Named Data changes can be more frequent than screen layout changes, the Forms Development subsystem provides a convenient editing facility using your default editor for Named Data.

A.10.5 References

VAX-11 FMS documentation set

A.11 Exercise 10 — Creating an Entry Form for an Indexed File

A.11.1 Objective

These exercises demonstrate how easy it is to create an indexed (ISAM) file from an ALL-IN-1 entry form.

A.11.2 Preparation

You should know how to use the FMS forms editor, be familiar with Named Data, and have a basic understanding of VMS forms libraries and form processing. You also need to copy a form using the Forms Development menu, and have forms library USER on your User Profile's form library line.

Copy the standard entry form using the CO option on the Forms Development menu your USER.FLB Form Library.

A.11.3 Exercise

Using the forms development subsystem, edit the screen layout of the form ENTRY in forms Library USER. Change the layout so that the second field is ten characters long. Change the title to be EXERCISE 11-ENTRY FORM. Name field 1 NAME and field 2 BIRTHDATE.

Use the FMS option to assign field attributes for all fields. Rename the fields to be NAME and DATE. Require a response in the NAME field, and use only uppercase letters for this field.

Next, change the Named Data for the form. The hard copy prompt should say: /HARD="Name and Date file". The .FILE data is as follows:

EXERCISE.DAT,NAME

This tells ALL-IN-1 that the file associated with the form is called EXERCISE.DAT and the keyword is the NAME field.

You should also change the Named Data for FIELD 1. You may clear the name and data for this field. You might wish to display a message to the user saying that the date field is optional. You can do this with the following:

The NAME line will be: DATE
The DATA line will be: /PRE= "DISPLAY This field is optional"

Next use the Assign Form Attributes option of FMS to rename the form EXERCISE.

Leave FMS, saving your changes.

You can now create the file named EXERCISE.DAT from the form EXERCISE by typing <CREATE EXERCISE on any ALL-IN-1 menu. Test the entry form (after creating the file) by using form EXERCISE (just type EXERCISE on an ALL-IN-1 menu).

A.11.4 Summary

This procedure creates a simple entry form with two fields. By using Named Data to specify the filename and key, the form is all that is needed to create a file with the same two fields in it. The fields in the file are the same length and have the same names as the fields on the form.

This facility is powerful for developing new applications. By creating an entry form for a new file, that file can be created quickly, and will be known to match the form field to field. Notice that if you change a field size or add a new field to an entry form, you will need to recreate the associated file.

A.11.5 References

VAX-11 FMS documentation set

A.12 Exercise 11 — Creating a Select Form

A.12.1 Objective

This exercise demonstrates how you allow a user to select records from the file you created with the entry form in Exercise 10. It is intended to make you more familiar with select forms.

A.12.2 Preparation

You must have completed Exercise 10, Creating an Entry Form for an Indexed File.

A.12.3 Exercise

Using the Forms Development menu, create a new form. When prompted for the form type, choose the standard SELECT form.

Once inside the FMS forms editor, use the Layout option to change the field descriptions to NAME and BIRTH DATE.

Use the Assign Field Attributes option to rename the input fields to NAME and BIRTHDATE.

Using the Data option, change the .TYPE Named Data to say:

```
Name .TYPE
SELECT FOR EXERCISE WITH .NAME = NAME
Name .TYPE
DO SEL_STYLE .NAME:20 .BIRTHDATE:20/LIST = 'EX14.SEL'
Name .TYPE
/STYLE = DISPLAY/PRE = 'DISPLAY Enter selection fields and press
<RETURN
```

Save your changes. Now type SELECT on any menu. Your select form displays, and you can enter selection criteria. These will be used to display records from the file you created in Exercise 10.

A.12.4 Summary

This exercise is designed to help you create a simple select form. Notice that you have to make screen layout and Named Data changes to enable the selection. In addition to the Named Data provided in the exercise, you may wish to add other form and field qualifiers in your custom applications.

A.12.5 References

ALL-IN-1 Office Menu Application Programmer's Reference

A.13 Exercise 12 — Modifying a Script

A.13.1 Objective

This exercise gives you an opportunity to see what can be achieved when using the script facility.

You should be familiar enough with scripts to decide whether this is a DO script or a SCRIPT script. You should also become familiar with the ALL-IN-1 operations performed in OA$LIB:FCEW.SCP.

It is also useful to look up the operations performed in FCEW.SCP to feel comfortable with what they do.

A.13.2 Preparation

Copy FCEW.SCP from OA$LIB to your default directory.

A.13.3 Exercise

Add an option to allow you to delete the contents of another folder after removing the contents of your wastebasket. Use a loop which clears out the folder and then asks you if you wish to do another.

Advanced exercise: Modify the script to perform the same task (including part one above) without using the CABINET JANITOR function. (Hint: use CABINET DELETE_DOCUMENT.)

A.13.4 Summary

This is an opportunity to do symbol processing and execute ALL-IN-1 functions in a DO script environment. Now you have had an opportunity to become familiar with what occurs when a script runs.

A.13.5 References

ALL-IN-1 Office Menu Application Programmer's Reference

A.13.6 Answers

The following script performs Exercise 1:

```
!FCEW.SCP
!Allows user to empty the wastebasket
GET #WASTE = OA$WASTEBASKET

.LABEL CLEAN_OUT

YESNO_PROMPT 'Do you really want to delete all documents
in the ' #WASTE ' [Y/N] ? \OA$FLD_STAY
.If OA$PROMPT_TEXT NES "Y" THEN .EXIT
GET OA$FUNCTION = "DISPLAY Emptying the " #WASTE " ...\FORCE"
GET DPTEMP = @#CURDOC
CABINET JANITOR #WASTE
.IF #DPTEMP:30 NES #WASTE THEN .GOTO EXIT_PROCEDURE
CABINET NEXT_FOLDER SPTEMP,@#CURDOC
.IF OA$STATUS = = 1 THEN .GOTO EXIT_PROCEDURE
CABINET SELECT,,,@#CURDOC

.LABEL EXIT_PROCEDURE

GET OA$FUNCTION = "DISPLAY" #WASTE " has been emptied\FORCE"
YESNO_PROMPT 'Do you want to empty another folder [Y/N]?'- \OA$FLD_STAY
.IF OA$PROMPT_TEXT NES "Y" THEN .EXIT
PROMPT 'Please enter the folder you wish to empty:'- \OA$FLD_STAY
.IF OA$FORM_TERMINATOR EQ 112 THEN .EXIT
GET #WASTE = OA$PROMPT_TEXT
.GOTO CLEAN_OUT
.EXIT
```

The answer to the advanced exercise 2 is:

Replace the following line in the solution to exercise 1:

```
CABINET JANITOR #WASTE
```

with the following lines:

```
.LABEL LOOP

CABINET NEXT_DOCUMENT #WASTE
.IF OA$STATUS = = 0 THEN .GOTO EXIT_PROCEDURE
CABINET DELETE_DOCUMENT
.GOTO LOOP
```

A.14 Exercise 13 — Creating a Shared Template

A.14.1 Objective

This exercise shows you how to create a template document to be shared by several users, and retrieve the template in the Word and Document Processing create option.

A.14.2 Preparation

You need the same privileges as the System Manager and have the OA$LIB: MANAGER form library in your User Profile form library field.

A.14.3 Exercise

This exercise allows you to create a document in your file cabinet that contains the typical text of a template. Give this document the name of your choice, but use the following sample data to create the actual template.

Date: <OA$DATE> <OA$TIME>

My name is <PROFIL.FULNAM[$USERNAME]>.
My ALL-IN-1 user name is <$USERNAME>.

The document I am currently editing is:
<OA$CURDOC> <OA$CURDOC_FOLDER> (<OA$CURDOC_TITLE>)

Press Gold F to file the document.

From the WP menu, enter FULLDB (this is the name of a form which makes all fields within the File Cabinet document visible). The information for the document you just created should be shown. Be sure to write down the name of the VMS file for your document. (You will need this information later.) Press RETURN or EXIT SCREEN to go back to the menu.

Enter SM to go to the System Manager menu. Select the FORMAT master with the Select (SEL) option. Now use the Create option. Type EXERCISE for the format name, and fill in the file name field with the VMS file name of the template document you created. Press RETURN to create the format entry.

Type WP to return to the Word and Document Processing menu. Now enter C to create a new document. Type the folder and title of your choice and use TAB to go to the Keyword field. Press TAB again. You now see the second half of the create document form set. Type EXERCISE in the Handling field and press RETURN.

You should now be editing a new document that has already been initialized with the format that you created previously in this exercise. Notice that the symbols and field names enclosed by angle brackets (< >) have been replaced by their values.

A.14.4 Summary

Templates, also known as boiler plates, are useful for creating multiple documents with similar formats. You have just created a template which can be shared by anyone who puts EXERCISE in the Handling field when they create a document.

A.14.5 References

ALL-IN-1 Office Menu User's Reference

A.15 Exercise 14 — Using DATATRIEVE

A.15.1 Objective

This exercise familiarizes you with the interface to callable DATATRIEVE. The interface allows an ALL-IN-1 user to enter DATATRIEVE (DTR) either interactively, or to have DTR perform fixed commands or procedures from menu selections. It also provides a means of storing the output of DTR in documents and mail messages.

A.15.2 Preparation

You must be in ALL-IN-1 and have DTR running on the system you are using. In addition, the procedure that linked ALL-IN-1 (LINKA1.COM) must be prepared to use DTR. Refer to the *Installation Guide* for further details.

To check to see if DTR is activated, type <DTR from any menu. If you get a DTR prompt, you have access to DTR. If you get an error message, you need to install DTR.

A.15.3 Exercise

If you are not at the DTR prompt, type <DTR from any menu. Once in DTR, you can use any DTR command except Guide Mode or the ADT facility.

For example:

```
DTR>                    Screen clears

DTR> SHOW ALL

Press <RETURN> to continue or EXIT SCREEN to exit

DTR>
```

NOTE: *If the logical DTR$STARTUP has been defined, a start-up command file will be executed when you enter DATATRIEVE for the first time.*

Because you are still in ALL-IN-1, your ALL-IN-1 context remains intact. In addition, ALL-IN-1 also preserves the DTR context when you leave DTR and return to ALL-IN-1. This allows you to go back and forth between ALL-IN-1 and DTR without losing your place. For example, if you are in DTR and leave to read some mail messages in ALL-IN-1, you can return to the same place in DTR after reading your mail messages.

Another way of reading mail while in DATATRIEVE is to press Gold I to display the Interrupt menu. From this menu you can perform several functions such as using an editor, the Desk Calculator, and the Time-Management facilities. An EXIT SCREEN from the Interrupt menu places the user back into DATATRIEVE.

If you want a DTR command to be executed and then have control return immediately to ALL-IN-1, include one or more DTR commands on the DTR function command line. Separate each command by a ";".

For example:

```
<DTR SHOW DOMAINS;SHOW RECORDS
```

Try some DTR commands. Note that if you have a domain defined for files used by ALL-IN-1, such as the User Profile or the Document File Cabinet, ready these commands with the SHARED clause. If you do not do this, DTR opens the file for exclusive access, denying access to ALL-IN-1. This occurs since DTR domains do not automatically close when you return to ALL-IN-1. To close all readied domains in DTR explicitly, use the FINISH command.

A.15.4 Summary

You now know how to access DTR from ALL-IN-1. You probably have additional ideas on how this interface can be used. For example, a command procedure in the subprocess can issue a series of DTR commands via WRITE OAMAILBOX.

Note that Callable DTR executes in the main process, not the subprocess. Any logicals referenced by DTR are those established in the main process. You could have two DTRs running at the same time: callable DTR in the main process, and the regular DTR in the subprocess. Although this is possible, it usually is not useful. Since the callable DTR is run only once (when ALL-IN-1 is started), it is much faster, as well as more convenient, if you are planning to go back and forth between DTR and ALL-IN-1.

A.15.5 References

VAX-11 DATATRIEVE documentation set

B

ALL-IN-1 Hard Copy

B.1 ALL-IN-1 Hard Copy

The ALL-IN-1 function call MODE C activates Hard Copy Mode. It is disabled by MODE S (for Screen Mode). ALL-IN-1 also activates Hard Copy Mode during initialization if your terminal is not a recognized DEC-CRT type. See Table B-1 for a list of recognized terminals that initialize automatically in Hard Copy Mode.

B.1.1 Intended Use

FMS forms do not display in Hard Copy Mode. ALL-IN-1 prompts for input to FMS fields in the format covered in this appendix.

Hard Copy Mode lets you use ALL-IN-1 from a non-DEC-CRT type terminal (for example, a hard-copy terminal such as a LA36 or a display terminal which is not compatible with FMS). You can also invoke this mode on a VT100 and similar display terminals when you want to bypass the FMS forms interface for performance reasons.

B.1.2 Terminal Support

ALL-IN-1 recognizes the terminals listed in Table B-1 and initializes in the associated mode when it runs from these terminals. You can reset the default modes listed in the table with the MODE function if the new mode is compatible with the terminal.

Table B-1 Supported Terminals

Terminal	Default Mode	Notes
PC350	Screen	
VT52	Hard copy	
VT100	Screen	
VT102	Screen	
VT125	Screen	
VT132	Screen	
VT220	Screen	<LK02 special keys>
VT222	Screen	<LK02 special keys>
VT240	Screen	<LK02 special keys>
VT242	Screen	<LK02 special keys>
GIGI	Screen	
LA32	Hard Copy	

ALL-IN-1 recognizes PC350 terminals as VT100s under Version 1.1 of P/OS. On PC350s running other versions of P/OS, enter the following sequence at the choice field prompt when ALL-IN-1 initializes in Hard Copy Mode:

```
CHOICE> <MODE T32{CR}
CHOICE> <MODE SCREEN {CR}{CR}
```

ALL-IN-1 then enters Screen Mode.

B.1.3 Function Keys

Since Hard Copy Mode can not assume the presence of an auxiliary keypad, it allows you to simulate keypad functions using the {ESC} key on the main keyboard. Thus:

{ESC} A-Z = Gold A-Z
{ESC} 0-9 = Keypad 0-9
{ESC} , = Keypad ,
{ESC} - = Keypad -
{ESC} . = Keypad .

For example, the normal EXIT SCREEN function is obtained by pressing {ESC} 0 (or CTRL/Z). In addition, the sequence

{ESC}{ESC}

displays the screen. The current contents of the fields display, but without video attributes.

If you use Hard Copy Mode with a VT1XX or VT2XX terminal to improve performance, the auxiliary keypad is still available, just as it is in Screen Mode, provided you entered with the terminal in application mode. However, ALL-IN-1 does *not* recognize the arrow keys.

B.1.4 Appearance on the Screen

Whenever the form changes, ALL-IN-1 displays a form identifier:

```
FORM)FMS-form-name
```

Whenever ALL-IN-1 requests input into a FMS field, it displays the following type of prompt:

```
)field-id [current-value])
```

where field-id is the FMS field name. If there is no current value in the field (for example, when field-attributes include Clear Before Input), then the square brackets are not shown. Also see the following section.

B.1.5 /HARD Qualifiers

You can use the /HARD form qualifier in two ways: as a permanent Named Data qualifier on the .TYPE line or as a dynamically appended qualifier in FORM function calls. Use the /HARD field qualifier in Named Data in conjunction with a particular field on a form.

If the .TYPE Named Data entry of a form includes a /HARD="prompt-text" qualifier, then ALL-IN-1 displays the specified text while in Hard Copy Mode. Otherwise, it displays the FMS form name. For example:

Named Data

```
1 Name .TYPE
ENTRY/MODE=UPDATE/KEY=USER/HARD="User Profile"
```

results in:

```
FORM)User Profile
```

When you specify /HARD on a field, the prompt text displays in a similar fashion. For example, if Named Data contains the following entry:

Named Data

```
1 Name CHOICE
/HARD="Select option EM,WP,DM,TM,US, or EX"
```

the following prompt displays when you enter that field:

```
)Select EM,WP,DM,TM,US, or EX)
```

When a field qualified by /HARD already contains a value, ALL-IN-1 displays that value in brackets after the /HARD prompt text:

```
)full username [Don Joe/Production Account])
```

If you only enter a terminator (for example, TAB, RETURN, BS), then ALL-IN-1 preserves the current contents of the field. Otherwise, any new input replaces the previous input. In contrast, Screen-Mode input is character-for-character overstrike.

B.1.6 Function-Key Entry

When you press an {ESCAPE} or Gold key (that is, a key that requires a second key to complete the terminator sequence), Hard Copy Mode waits for you to complete the key sequence.

When you exit the field normally (TAB, RETURN, BS, and so on) ALL-IN-1 preserves the new value.

B.1.7 Field-Full Condition

If you fill a field completely and the next key you press is not a terminator, then the cursor returns to column 1 and the terminal beeps at each character you enter, until you enter a terminator. This corresponds to the FMS Field Full beeped message.

C

Data Sets

C.1 Overview

Chapter 1 covers the separation between form and function that underlies the VAX Information Architecture. ALL-IN-1 organizes its information into *data sets*, internal constructs which not only keep this separation consistent throughout the system but allow you to easily combine different types of information in certain situations.

For example, when you call the FORM function, ALL-IN-1 manages the form and the information that displays through it by creating a data set that connects the static screen picture to the volatile information stored in Named Data.

Classes of data sets are:

- **Data**

  An entry form and its related indexed file is a data-class data set in which the FMS fields on the form generally correspond one-to-one to RMS fields in the file. The File Cabinet is also a data-class data set, one that comprises several data files. Most of these File-Cabinet data files are specially formatted and not accessible through entry forms.

Special data sets, which are covered in the following sections, come under the data class.

- **Form**

  When you display an FMS form, ALL-IN-1 builds a data set that represents the video characteristics of the form, the layout of the fields, and information that you associate with the fields in Named Data. Some forms, namely entry forms, may also be associated with data-class data sets, as described above.

- **Text**

  Text files, which are unstructured data, are treated as text data sets. ALL-IN-1 looks at text data sets line by line and processes them in the context of a standard format.

C.2 Special Data Sets

ALL-IN-1 provides several special data-class data sets for field validation, file searches, and other limited tasks. These data sets are not associated with either data files or forms and are instead accessed through *data set references*, or DSRs. DSRs often look like special symbols or special GET destinations, but they are more powerful.

The DSRs that represent special data sets are covered in the following sections.

C.2.1 OA$DATE

Use this DSR in validation expressions when you do not want the users to move off the field until they have entered a valid date.

The date may be expressed as:

dd-mmm-yyyy

OR

dd/mmm/yy

When you qualify a field with /INVALID=OA$DATE, then the user *cannot* enter a value in this format.

OA$DATE can only be used with /VALID and /INVALID.

C.2.2 OA$DIR

OA$DIR provides a way to search, validate against, and list file names from a directory. You can use this DSR in validation expressions, in FOR function calls, and in GET function calls. The syntax varies depending on where you use it.

C.2.2.1 In Validation Expressions.

Use OA$DIR in a validation expression when you want to ensure that a user enters a valid RMS file specification in a field. OA$DIR in this context has the following syntax:

```
OA$DIR:default-file-spec-symbol[.field-name]
```

where

default-file-spec-symbol — is an ALL-IN-1 symbol-table symbol or quoted string which specifies any valid RMS default file specification. You can use wildcard characters in the file specification.

field-name — is an optional keyword that identifies which part of the file-spec to validate against. When you omit the keyword, %WHOLE is assumed. OA$DIR recognizes the following keywords:

- %WHOLE — Retrieves/validates against the fully parsed file name
- %NEXT — Retrieves/validates against the next file name in the directory
- NODE — Retrieves/validates against the node name
- DEV — Retrieves/validates against the device
- DIR — Retrieves/validates against the directory
- NAME — Retrieves/validates against the file-name root
- TYPE — Retrieves/validates against the file-type extension
- VER — Retrieves/validates against the file version
- ALLBUTVER — Retrieves/validates against the entire file specification without the semicolon and version number

For example:

```
/VALID = OA$DIR:"OA$LIB"
```

this expression validates the contents of the qualified field against the expanded file-spec for logical name OA$LIB.

```
/RECOGNITION = OA$DIR:$DATA_FILES.NAME
```

If $DATA_FILES has the value "OA$LIB:*.DAT", then pressing Gold L or Gold S in an empty field displays the file-name root for each of the data files in directory OA$LIB.

C.2.2.2 With the FOR Function. Use OA$DIR with the FOR function when you want to execute one or more DO subfunctions for each file in a given group of RMS files. The syntax is:

```
FOR OA$DIR:file-spec-symbol DO function [\ \function...] ]
```

where file-spec-symbol is an ALL-IN-1 symbol that represents a valid RMS file specification; wildcard characters are allowed. For each file that meets this specification, FOR executes the DO function list. For example:

```
FOR OA$DIR:"*.LIS" DO PROMPT .%WHOLE " is here; press RETURN"
```

In this call, FOR collects all files in the current default directory with the extension .LIS and prompts the user with the full file name. All versions of each file are collected by default: to process only the most recent version of a given file, include a semicolon (;) at the end of the file extension or specify "root.ext;0". Another example follows:

```
FOR OA$DIR:#LISTFILE ";0" DO GET OA$FUNCTION = "MERGE " .NAME ".BLP,"
  .NAME ".OUT," .%WHOLE
```

#LISTFILE contains some RMS file specification that identifies one or more MERGE list-file names. #LISTFILE, for example, may have the value *.LIS, in which case the latest versions of all files in the default directory with the .LIS extension are treated as MERGE list files and processed by MERGE through the DO call. The current file's file-name root, identified by the special field-name .NAME, is used to build the rest of the MERGE call.

C.2.2.3 With the GET Function. With GET, use OA$DIR as a source symbol to identify a single file name. OA$DIR returns the file name, if any, that matches the given file-spec information. If you supply one or more destination symbols, OA$DIR gives them the value of the file name. If you call OA$DIR by itself, GET processes it like other single-parameter GETs.

The syntax is:

```
OA$DIR:default-file-spec-symbol.field-name[file-spec-symbol]
```

where:

default-file-spec-symbol — is an ALL-IN-1 symbol which specifies any valid RMS default file specification. You can use wildcard characters in the file specification.

field-name — is any one of eight keywords that identify which part of the file-spec to retrieve.

file-spec-symbol — is an ALL-IN-1 symbol which identifies one or more parts of the file-spec to search for. The square brackets are required, and wildcard characters may be used.

Unlike the default file specification, the bracketed file specification need not be a full RMS file specification. By default, GET treats the bracketed file specification as a file-name root. However, it treats any punctuation marks such as double colons, colons, periods, and semicolons in the bracketed file specification as normal file-spec delimiters. That is, specifying AB between the brackets causes OA$DIR to search for files that match the wildcard specification AB.*;*, while specifying A.B causes OA$DIR to search for file A.B, where A is the file-name root and B is the file-type extension.

NOTE: Although you may omit the file-spec symbol when you specify OA$DIR in the FOR function and in validation expressions, you must supply all parts of this DSR with the GET function.

The default file specification and the bracketed file specification together determine which files meet the selection criteria. For example:

```
<GET OA$DIR:"*.DAT;*".%WHOLE["DOC%%%"]
```

This expression returns, on line 24, the expanded name of the first .DAT file in the default directory with a six-character file-name root that begins with DOC; for example:

```
SYS$:[JONES.OA]DOC001.DAT;1
```

The same results would be obtained by specifying:

```
<GET OA$DIR:"DOC%%%.*;*".%WHOLE["*.DAT;*"]
```

where the default file specification and the bracketed file specification are simply reversed. The default file specification should usually provide context information, such as a directory or device name to search, while the bracketed file specification should pinpoint the file or files to return. To save the file-name root of the first file in your default directory in symbol #FIRST, simply specify full wildcarding for both file specifications:

```
<GET #FIRST = OA$DIR:"*.*;*".%NAME["*"]
```

Use the %NEXT keyword to get the next file name as follows:

```
<GET OA$DIR:"*.*;*".%NEXT[#FIRST]
```

C.2.3 OA$DIR_PARSE

Use OA$DIR_PARSE to validate against a VMS RMS file specification. You can only use this DSR in validation expressions.

C.2.3.1 In Validation Expressions. Use OA$DIR_PARSE in a validation expression to ensure that a user enters a valid RMS file specification in a field. OA$DIR_PARSE differs from OA$DIR because it checks the specification for syntatical correctness and not for existence in a VMS directory. Thus, use OA$DIR_PARSE in create operations where a new file is generated.

OA$DIR_PARSE has the following syntax:

```
OA$DIR_PARSE:default-file-spec-symbol
```

where

default-file-spec-symbol — is an ALL-IN-1 symbol-table symbol or quoted string which specifies any valid RMS default file specification. You may not use wildcard characters in the file specification.

For example:

```
/VALID = OA$DIR_PARSE:"OA$LIB"
```

This expression validates the contents of the qualified field against the expanded file-spec for logical name OA$LIB.

C.2.4 OA$EDITOR

Use OA$EDITOR in a validation expression to validate an editor style.
/HARD=Editor-style/VALID="OA$EDITOR"' ;

C.2.5 OA$MAIDES

Use OA$MAIDES in a validation expression to provide recognition against the valid ALL-IN-1 mail destinations. The supported destinations are:

- ALLIN1
- VAXMAIL

Since MAIL invokes an alternate message transport system if it encounters an unknown destination, you do not necessarily have to restrict a user to entering supported destinations in a field. That is, use /VALID="OA$MAIDES" on a field only when you do not want to create and support an alternate message transport system. You would generally use OA$MAIDES with /RECOGNITION.

C.2.6 OA$MAIL_ADDRESS

Use OA$MAIL_ADDRESS in a validation expression to ensure that the user enters a mail address in a field that:

- Contains special prefixes or suffixes that identify a special address
- Is contained in the User Profile or some other list of valid mail recipients

C.2.7 OA$MAILPRIO

OA$MAILPRIO is a specialized DSR that validates against an internal table to ensure that the user enters a valid MAIL priority in the qualified field. The valid priorities are:

Express
First_class
Second_class

C.2.8 OA$TABLE

Use OA$TABLE to build a specialized table of entries to validate against. The syntax is:

```
OA$TABLE:"entry1,entry2..."
```

Enclose the list in quotes, and separate individual list items by commas. For example:

Named Data

```
1 Name SELDIR
/GET = $DIR/VALID = OA$TABLE:"PERSONAL,ALL-IN-1,CORPORATE"

2 Name PMODE
/GET = $CAL_PMODE/PUT = $CAL_PMODE/VALID = OA$TABLE:"S,T,B"

3 NAME PTRTYP
/VALID = OA$TABLE:"LP11,LA100,LA50,LQP02"/PUT = #PTRTYP

4 Name AI_PRIORITY
/INVALID = OA$TABLE:"0, + ,-"
```

The user must fill fields SELDIR, PMODE, and PTRTYP with one of the three table values listed for those fields, while field AI_PRIORITY cannot contain any of the three special characters specified.

C.2.9 OA$YESNO

Use this special DSR with /VALID to ensure that the user enters either the word yes or the word no. If you qualify a field with /INVALID=OA$YESNO, ALL-IN-1 treats anything but yes or no as a valid answer.

For example:

```
1 Name ANSWER
/VALID = OA$YESNO
```

C.2.10 OA$YN

Use this special DSR with /VALID to ensure that the user enters either the character y or the character n in the qualified field. The inverse is true if the field is qualified with /INVALID=OA$YN. If you qualify a field with /INVALID=OA$YN, ALL-IN-1 treats anything but y or n as a valid answer.

For example:

```
1 Name ANSWER
/VALID = OA$YN
```

C.3 Calendar Validation Data Sets

The CALENDAR function employs several validation data sets that validate dates, times, and user information.

C.3.1 CAL$ATTENDEE

This data set validates meeting attendees.

C.3.2 CAL$CALENDAR

This data set can be used to validate the date and time of meetings for a single user.

For example:

```
1 Name DATE
GET DATE = CAL$CALENDAR.START_DATE:11[OA$SEL_KEY]
```

C.3.3 CAL$END_TIME_CHECK:key

This data set validates an end time against the start specified by key.

For example:

```
1 Name ETIME
/GET = #CAL_MTNG_ETIME/VALID = CAL$END_TIME_CHECK:"STIME,ETIME"
```

C.3.4 CAL$MEETING

This data set validates the date and time of meetings for the system.

C.3.5 CAL$MTG_TIME_CHECK

This data set validates the date and time against system time.

For example:

```
1 Name STIME
/GET = $AM_SELECT1/VALID = CAL$MTG_TIME_CHECK
```

C.3.6 CAL$SET_DATE

This data set validates a date command and sets the current calendar date.

For example:

```
1 Name DATE
/GET = OA$TM_DATE/VALID = CAL$SET_DATE
```

C.3.7 CAL$SET_OWNER

This data set validates a calendar owner and sets up the User Profile fields for that owner and makes sure the real user has access.

For example:

```
1 Name USER
/VALID = CAL$SET_OWNER/GET_SAVE = OA$TM_OWNER
```

C.3.8 CAL$SET_SECOND

This data set validates access to a second user's calendar for the two calendar display.

For example:

```
1 Name USER
/PUT = #CAL_SECOND/VALID = CAL$SET_SECOND
```

C.3.9 CAL$TIME_CHECK

This data set validates a time.

For example:

```
47 STARTH'
/HARD = Work-day: Start-hour?/VALID = "CAL$TIME_CHECK"

1 Name TIME
/HARD = "Time of Meeting"/VALID = CAL$TIME_CHECK:"Time"
```

C.4 Calendar Scrolling Data Sets

CALENDAR provides scrolling data sets that allow you to maintain Time Management files through scrolled regions.

C.4.1 CAL$SCROLL__BOTH

This special scrolling DSAB scrolls both schedule and reminders for a day.

For example:

```
1 Name DESC
/SCROLL = 96,,1,CAL$SCROLL__BOTH
```

C.4.2 CAL$SCROLL__CONFLICTS

This special scrolling DSAB scrolls conflicts in a day's schedule.

For example:

```
1 Name DESC
/SCROLL = 96,,1,CAL$SCROLL__CONFLICTS
```

C.4.3 CAL$SCROLL__CUSTOM

This special scrolling DSAB scrolls a day schedule in a customized format. See SEL__DISPLAY for formatting instructions.

C.4.4 CAL$SCROLL__DAY

This special scrolling DSAB scrolls a day's schedule.

For example:

```
1 Name DESC
/SCROLL = 96,,1,CAL$SCROLL__DAY
```

C.4.5 CAL$SCROLL__TWOCAL

This special scrolling DSAB scrolls two calendars side by side.

For example:

```
1 Name DESC
/SCROLL = 1440,,1,#CAL__START,CAL$SCROLL__TWOCAL
```

C.4.6 CAL$SCROLL__WEEK__REM

This special scrolling DSAB scrolls a week reminder.

For example:

```
1 Name DESC
/SCROLL = 1440,,#CAL__START,CAL$SCROLL__WEEK__SCH
```

C.4.7 CAL$SCROLL__WEEK__SCH

This special scrolling DSAB scrolls a week schedule.

For example:

```
1 Name DESC
/SCROLL = 1440,,#CAL__START,CAL$SCROLL__WEEK__SCH
```

C.5 General Scrolling Data Sets

C.5.1 SCROLL$FUNCTION

This general scrolling DSAB allows you to customize scrolling.

C.6 Data Set Statistics

When you invoke a task that ALL-IN-1 performs through a data set, it first checks whether the data set is open. As covered under the DUMP__CACHE and OA$DSA__CACHE__STATUS functions, ALL-IN-1 keeps recently and/or frequently used data sets in physical memory to enhance performance.

When you call OA$DSA__CACHE__STATUS, it builds and displays a table of data-set-related statistics. Table C-1 illustrates a typical OA$DSA__CACHE__STATUS display and explains these statistics.

Table C-1 Data Set Statistics (OA$DSA_CACHE_STATUS output)

Numbers at the top of each column correspond to notes at the end of the table

1	2	3	4	5	6	7	8	9	10	11	12	13
DSAB NAME	O P N	D Y N	C S H	P R V	P O N	C R O	REF CNT	DSAB TYPE	NUMB OF FIELDS	MAX RECORD SIZE	DSAB ADDRESS	RAB ADDRESS
** DATA **												
CAL$ATTENDEE	1	1	1	0	0	1	0	0	0	0	005AF548	005AF668
OA$CABFIFOS	1	1	0	0	0	1	1	3	0	0	005943E8	00057DAC
CAB$PENDING	1	1	0	1	0	1	1	0	8	2000	00549E38	00057DCA
CAB$SHARE_COUNT	1	1	0	1	0	1	1	0	36	2000	0053AD46	0005D7AC
CAB$SDAF	1	1	0	1	0	1	1	0	36	2000	00584E83	00057454
CAB$PDAF	1	1	0	0	0	1	1	0	36	2000	00592EA3	00057ADC
DOCDB	1	1	0	0	0	1	1	0	23	500	005953F9	00057EBD
CAB$	1	1	0	0	0	1	2	0	59	2000	00596F0A	00057FCE
NIENTR	1	1	1	0	0	1	1	0	2	62	005970AB	000570DF
OA$LOCAL_MEMORY	1	1	0	0	0	1	1	3	2	512	005981AC	000571E0
PROFIL	1	1	0	0	0	1	1	0	66	950	005992BD	000572F1
DSAB NAME	O P N	D Y N	C S H	P R V	P O N	C R O	REF CNT	DSAB TYPE	NUMB OF FIELDS	MAX RECORD SIZE	DSAB ADDRESS	RAB ADDRESS
** TEXT **												
DSAB NAME	O P N	D Y N	C S H	P R V	P O N	C R O	REF CNT	DSAB TYPE	NUMB OF FIELDS	MAX RECORD SIZE	DSAB ADDRESS	PREV DSAB
** FORM **												
TM	1	1	1	0	0	0	1	0	5	256	0059238E	0005ADC7
DEFAULT	1	1	1	0	0	0	1	0	1	256	0054E38E	00000000

Notes

1 DSAB NAME — The name of the data set; DSAB stands for data set access block.

The names of data DSABs correspond either to the entry form name (for example, PROFIL) or to a special name that identifies the data set (that is, the data set reference, such as CAB$).

Text DSABs are usually active only within the context of an edit session, and OA$EDITBUFFER displays here if you call OA$CACHE_STATUS from within the default editor.

The names of form DSABs correspond to form names.

2 OPN — Indicates whether the data set is open. By definition, this is always 1 for cached data sets.

3 DYN — Indicates whether the DSAB is dynamically allocated.

4 CSH — Indicates whether the DSAB can be left open while not being used.

5 PRV — Indicates whether privileges may be elevated in order to access a file associated with this data set.

6 PON — Indicates whether SYSPRV was turned on to access a file associated with this data set.

7 CHO — Indicates whether a user can get a Choices list with this data set.

8 REF CNT — The current DSAB reference count: the number of times you have referenced this data set. The caching algorithm uses this figure.

9 DSAB TYPE — Corresponds to the DSAB class.

10 NUMB OF FIELDS — Number of fields in the DSAB. For data DSABs, this usually corresponds to the number of RMS fields in the file, and for form DSABs, this usually corresponds to the number of FMS fields on the form.

11 MAX RECORD SIZE — Maximum DSAB record size.

12 DSAB Address — Memory address of DSAB.

13 RAB Address — For data DSABs, this is the memory address of the RMS record access block for the DSABs data file.

OR

PREV DSAB — For form DSABs, this is the memory address of the previous form. Each form DSAB remembers the address of the previous form in this way; several functions, including OA$FLD_EXIT and CLOSE_PRIOR, use this information.

D

Mnemonics for Script Input

Appendix D is an on-line table that lists mnemonics for script input. The table provides information about the character name, the encoded value, the character set and the character type. These mnemonics allow special keys, control keys and keypad sequences to be referenced by name. They are used when:

- A Script returns input to ALL-IN-1.
- A Script uses the .JUDGE command to test the input entered by a user.
- An .IF statement compares the value of OA$SCRIPT_PSIB, OA$SCRIPT_TEXT, or OA$SCRIPT_TERM.

The Appendix is included in your ALL-IN-1 directory, OA$LIB, under the file name SCPTABLE.TXT. You can access this on-line table in several ways:

- By scrolling the file on your screen. You must first set your terminal width to 132 characters. For example:

```
$ SET TERMINAL/WIDTH = 132
$ TYPE OA$LIB:SCPTABLE.TXT
```

- By printing the file. For example:

  ```
  $ PRINT OA$LIB:SCPTABLE.TXT
  ```

- By importing the file into your ALL-IN-1 File Cabinet using the RV option (Receive a document from VMS) on the Document Transfer menu.

Glossary

ACMS

See *Application Control and Management* System.

account

The allocation of ALL-IN-1 facilities to each user. A user must have an account to use the *system*. Each user has a separate account, identified by a user name and a *password*.

action item

An activity the user schedules to be completed by a specific date.

Action Items

An *option* of the *Time Management* menu.

Action Item List

A list of outstanding actions, a task list, which are not scheduled in the *calendar*.

alphanumeric

Consisting of letters, numbers, or a combination thereof.

application

An *operation* or procedure using specific equipment or programs for a defined task, usually not part of the basic *VAX/VMS operating system.* For example, *Electronic Messaging* and *Desk Calculator* are ALL-IN-1 applications.

Application Control and Management System (ACMS)

A VMS layered product that simplifies management of on-line applications.

appointment

An activity the user schedules for a specific time and date but just on that user's calendar.

APR

Application Programmer's Reference; this document.

Arrow key

One of four keys that may be used to move the *cursor* up, down, left, or right.

ASCII

The American Standard Code for Information Interchange, a set of 8-bit binary numbers representing the alphabet, punctuation, numerals, and other special symbols used in text representation and communications protocol. ALL-IN-1 uses ASCII coded characters when communicating with other computer systems.

attachment

A *document* or *message* that is sent or forwarded as part of another document or message.

auto dial

A *feature* used when setting up a *communications* line to a *host* computer using a *modem*. The user types the host's phone number at the *keyboard* and ALL-IN-1 dials the number.

auto forward

An *option* of *Electronic Messaging* whereby the user can have incoming *mail* automatically sent to another *system* or account on the same system.

auto reply

An *option* of *Electronic Messaging* whereby the user can have a reply to incoming *mail* automatically sent to the originator of the *message*.

back up

To move the *cursor* toward the top of a *document*.

backup

An additional copy of documents or *software* made for safekeeping in case the original is accidentally damaged or destroyed.

batch processing

Method of processing in which all commands to be executed by the *operating system* and any *data* to be used as *input* to the commands are placed in a *file* and submitted to the *system* for execution.

block

A measurement of storage on a *disk* or a *diskette* equivalent to 512 characters.

Boolean Expression

Search instructions for selecting records from a *data set.* A Boolean expression is a combination of fields in the data set, relational operators, ALL-IN-1 symbols, and Boolean operators.

Used in FOR function calls, validation expressions, and Script .IF calls.

Boolean Operator

A keyword that links parts of *Boolean expression.* The valid Boolean operators are AND, OR, and NOT.

bottom of document

The end of a *document.* Pressing Gold B in the WPS-style editor moves the user directly to the bottom of the document.

buffer

An area used to store *text* temporarily. For example, when the user presses the CUT key, selected text moves to a *paste buffer*, also known as a paste area, where it is stored until it is needed. (See also *cut* and *paste.*)

Business Applications

The ALL-IN-1 *subsystem* that contains applications that are specific to a user's business or profession.

Calculations

The ALL-IN-1 *subsystem* that allows the user to access the *Desk Calculator*.

Calculator Mode

The *operation mode* in the *Desk Calculator* that allows the user to perform simple arithmetic computations.

calendar

See *Time Management*.

calendar form

An ALL-IN-1 *form* that displays a *calendar* of the current month indicating scheduled appointments, meetings, reminders, and so on.

carriage return

An invisible *character* that forces all following *text* to the next line. This term is also known as RETURN. A carriage return character can be either system-supplied or operator-supplied and usually is not displayed on the *screen*.

CBI

See *Computer Based Instruction*.

CDD

See *Common Data Dictionary*.

center

To position a line of *text* an equal distance from the left and right margins. Produced by pressing Gold C. The user also may specify a typing position around which a typed line is to be centered.

centering point

A point in a *ruler*, indicated by C, around which the *text* on that line is centered.

character

A single letter, composite character, digit, or *symbol*, produced by pressing a key on the *main keyboard.*

Certain invisible markers that take up the same amount of storage space in a *document* as a single letter. For example, a *carriage return.*

character transmission (CX)

A *communications option* that transmits one *character* at a time among the *keyboard*, the *screen*, stored documents, and another computer, called the *host.*

choice

An ALL-IN-1 feature that lets the user enter a *word*, or part of a word, into a *field.* When the user presses Gold L or Gold S, ALL-IN-1 lists all entries beginning with or containing the characters the user entered.

command

An instruction typed by the user on the *keyboard* or included in a command procedure. A command asks the *software* to perform some predefined *operation.*

Common Data Dictionary (CDD)

The cornerstone of the *VAX* Information Architecture. A collection of *data* about data used by *DATATRIEVE* and other *VMS data base* management systems.

communications

The capability by which ALL-IN-1 sends and receives characters or documents to and from another computer *system*. The prefix tele- is sometimes used to indicate transmission over a longer distance, for example, by a telephone link.

Communications

The ALL-IN-1 *subsystem* that allows the user to communicate with other users on the *computer network* and to transfer documents between systems.

Communications Control

The ALL-IN-1 *subsystem* that allows the user to connect to and use systems other than the *system* running ALL-IN-1.

Computer Based Instruction (CBI)

Lessons the user can *select* from the *Main Menu* and complete at the user's own pace at the *terminal*. The lessons are presented in a step-by-step *format*.

Lessons are also available through the TR *option* from many of the *subsystem* menus.

computer network

An interconnection of computer systems, terminals, and *communications* facilities.

connection

A link from ALL-IN-1 to another computer *system*, such as another ALL-IN-1 system, enabling the systems to communicate with each other.

control document

An ALL-IN-1 *document* that contains commands to automate *Communications Control* procedures.

control key

The CTRL key on the *main keyboard* that, when pressed along with another key, performs a predefined function. For example, holding down the CTRL key and pressing W will cause ALL-IN-1 to *restore* the *terminal screen.*

copy

To make a duplicate of a *document*, *folder*, or VMS *file.*

create

An option that causes the *system* to know that a new *document*, *folder*, or other item is to be produced.

cross-file

The ALL-IN-1 option that allows the user to assign more than one *folder name* to a *document* or group of documents.

Current Action Item Block

On the *Action Items menu*, that portion of the *screen* that displays the date, class, and item of the *action item* that is available for editing, reading, printing, and so on.

Current Appointment Block

On the Appointments *menu*, that portion of the *screen* that displays the date and time of the appointment that is available for editing, reading, printing, and so on.

Current Document Block

On the *Word and Document Processing menu*, the portion of the *screen* that displays the name of the *folder* and the title and number of the *document* that is available for editing, printing, deleting, and so on.

Current Entry Block

On the Corporate Phone Directory *menu*, the Personal Phone Directory menu, and the ALL-IN-1 Phone Directory menu, the portion of the *screen* that displays the name, business telephone number, and personal telephone number of the entry that is available for editing, reading, printing, and so on.

Current Form Block

On the *Forms Development menu*, the portion of the *screen* that displays the name of the form *library* and the name of the form that is available for editing, printing, reading, and so on.

Current Meeting Block

On the Meetings *menu*, the portion of the *screen* that displays the date and time of the meeting available for editing, reading, printing, and so on.

Current Message Block

On the *Electronic Messaging menu*, the portion of the *screen* that displays the name of the *folder* and the title, number, and status of the *message* that is available for editing, printing, sending, and so on.

Current Module Block

On the *Programming Functions menu*, the portion of the *screen* that displays the name of the *folder* and the title and number of the *module* that is available for editing, printing, and so on.

Current Reminder Block

On the Reminders *menu*, that portion of the *screen* that displays the date and number of the reminder that is available for editing, reading, printing, and so on.

Current Task Block

On the To-Do List *menu*, that portion of the *screen* that displays the date and number of the To-Do List task that is available for editing, reading, printing, and so on.

cursor

A movable, flashing pointer on the *terminal screen* showing where the next *character* typed will appear. Some terminals *display* the cursor as an underline, some as a rectangular block.

cut

To remove a section of *text* from a *document* by using the CUT key. The text is placed into a *paste buffer* and can be reinserted into a document by using the PASTE key. See *paste*.

CX

See *character transmission*.

data

Facts, numbers, letters, and symbols stored in the computer *system*. Data can be thought of as the basic elements of information used, created, or otherwise processed by an *application program*. Examples of data are employee names, weekly deductions from salary, projected sales, and fuel consumption.

data base

A collection of organized *data* from which, for example, a computer generates reports or mailing lists.

Data Set

A collection of *data* that is defined in terms of the user interface used to access it. For example, a data file and the *entry form* that describes it together are a data set; the data set name is the name of the entry form.

Some special data sets cannot be described by entry forms. In these cases, the *Data Set Reference*, or *DSR*, that is used to access the data set is the only user interface.

Data Set Reference (DSR)

A multipart *symbol* that retrieves information from a data set. A DSR commonly consists of an entry form name, an *FMS* field name, and a bracketed value which indexes a record in the data set.

DATATRIEVE

A *data* management language for defining, storing, updating, and displaying records from *RMS* data files. DATATRIEVE is a trademark of Digital Equipment Corporation.

Date and Time

ALL-IN-1 displays the current date and time when the user enters the *system.*

An editor function that inserts the current date and time into the *document* being edited.

DCL

See DIGITAL Command Language.

DCLMAILBOX

The *VMS mailbox* that ALL-IN-1 creates to *send* information from the main process to the *subprocess.* Used by command procedures and programs running in the subprocess.

DECdx

Software that allows the user to transfer documents between a computer *system* and a DECmate.

DECdx Index

The list of documents that were transferred from a DECmate.

default

The *account, device,* or setting ALL-IN-1 automatically uses unless the user specifies different values. For example, the first *ruler* ALL-IN-1 displays when the user creates a *document* is the default ruler.

Information that the *operating system* assumes as *input* if the information is not explicitly provided.

delete

To remove a *character* or *word* at or to the right of the *cursor*. Typically, delete keys are used to remove previously typed *text* from the *document*.

To remove a document, *file*, or *directory* from storage on a *disk* or *diskette*.

An ALL-IN-1 option on most *subsystem* menus used to remove the current item.

delivery receipt

A *mail message* sent by the ALL-IN-1 *system* to the originator of the message stating that the message has been sent to the specified addressees.

Desk Calculator

The ALL-IN-1 *application* that lets the user use the *keypad* as a pocket calculator. This application also lets the user perform more complex computations when in *Formula Mode*.

Desk Management

The ALL-IN-1 *subsystem* that lets the user manage daily office activities. The user can maintain a daily *calendar*, *display* the *date and time*, access both a personal and corporate phone *directory*, and compute figures with an *on-line* calculator.

device

The general name for any peripheral *hardware* connected to the *processor* that is capable of receiving, storing, or transmitting *data*; for example, printers, terminals, and *disk* drives.

DIGITAL Command Language (DCL)

High-level commands used to execute tasks in *VMS*.

DIGITAL Standard Runoff (RUNOFF)

A *document* formatter used to prepare and highlight printed documents.

direction key

Any key that changes the direction in which the *cursor* moves, for example, BACKUP, ADVANCE, Left Arrow. (See also *Arrow key*.)

Directories

The ALL-IN-1 *subsystem* that allows the user to access the phone directories, the *Nickname Management* Subsystem, and the *Distribution List Management* Subsystem.

directory

A *VMS* file that is used to locate other files in storage. ALL-IN-1 is run from a *library* directory, and its *data* files are kept in a data directory. Each user is assigned a personal directory.

disk

A storage *device* consisting of one or more rigid, flat, circular plates housed in a plastic cartridge.

diskette

A flexible, flat, circular, magnetic medium permanently housed in a black plastic envelope. Also known as a floppy *disk*.

display

The *screen* of a video *terminal* or the images that are shown there. Also known as the CRT or cathode ray tube.

display a ruler

To *command* ALL-IN-1 to show the current *ruler* on the *screen* and allow its settings to be changed.

Display Register

An area on the *Desk Calculator form* into which the numbers the user enters and the results of the computations are placed.

distance key

Any one of the keys on the *keypad* that defines the distance the cursor can move in one *keystroke.*

Distribution List

A roster of people to whom the user frequently sends messages.

document

A *file* of *text* identified by a title and number and stored on a *disk.* One document typically contains one typing job, such as a memo, letter, or report. The document is the basic unit of the ALL-IN-1 filing system and is always filed in a *folder.*

Document Attributes

Information about a *File Cabinet document.* Individual attributes are stored either as fields in OAUSER:DOCDB.DAT or as special records in one of the document attributes files, OAUSER:DAF.DAT or OADATA:DAF.DAT.

document destination

A *field* on the Printing Document *entry form* that determines which printer will *print* a selected *document.*

document number

A number ALL-IN-1 assigns to a new *document* when it is created.

document title

The name assigned to a *document* by the user who created it. Use this title, or the *document number*, to refer to the document.

Document Transfer

The ALL-IN-1 *subsystem* that allows the user to move documents between systems.

draft printer

Any printer that produces documents of lesser quality than a *letter quality printer.* Typically used for internal documents for which type quality is not a major factor.

DSR

See *DIGITAL Standard Runoff.*

See *Data Set Reference.*

edit

An ALL-IN-1 option that lets the user make changes to the current item. To make changes to a *document*, *message*, and so on.

Editor Menu

A *menu* containing *options* relevant to editing. Invoked by pressing Gold M while creating or editing a *document.*

EDT Editor

DIGITAL's standard editing utility. A *program* that lets the user enter new information or correct existing information in ALL-IN-1 documents or *VMS* files.

EIA

See *Electronic Industries Association.*

Electronic Industries Association (EIA)

A standards organization specializing in the electrical and functional characteristics of interface equipment.

Electronic Messaging

The ALL-IN-1 *subsystem* that allows messages and documents to be sent to and received from other users on the same or remote computer systems.

embed

To insert a *ruler* into a *document* by pressing *RETURN* after displaying the ruler.

entry form

An ALL-IN-1 *display* composed of one or more related fields that *prompt* the user for information about a selected *option.* The user then enters information into the various form fields.

erase

To remove characters from the *text*. Use the rubout keys to remove text to the left of the *cursor*. Use the *delete* keys to remove characters at and to the right of the cursor.

error message

Text displayed when an incorrect response is typed. The error message explains the problem and indicates what to do next.

Exit

The *option* on the *Main Menu* that lets the user leave ALL-IN-1.

EXIT SCREEN

The ADVANCE (zero) key on the *keypad* that cancels the task being performed and returns the user to the last *menu* used.

facility

A part of ALL-IN-1 code that performs a specific task. Facility is an internal concept, but thinking of ALL-IN-1 in terms of facilities is useful when trying to sort out individual components.

FDL

See *File Definition Language*.

field

An area in a *form* that can change because of user *input* into the form or because of ALL-IN-1 actions. Fields are given attributes with the *FMS* Forms Editor and are further defined with ALL-IN-1 field qualifiers.

field name

A label enabling both the user and the *system* to uniquely identify a particular *field.*

figure

A diagram or pictorial illustration.

file

To save a *document,* on a *disk,* for future use by pressing Gold F when the user has finished creating or editing the document.

A collection of logically related records or *data* treated as a single item and stored on a magnetic medium, such as a *disk.*

File Cabinet

The ALL-IN-1 *subsystem* that lets the user move documents among folders.

The place where ALL-IN-1 stores documents and messages.

File Definition Language (FDL)

A *VMS* utility used to create data files.

filename

The sequence of *alphanumeric* characters assigned by the user to identify a *file* that can be read by both the *system* and the user.

filing

The act of storing the *data* or *document* in the electronic *File Cabinet.*

FMS

See *Forms Management System.*

folder

The unit of the ALL-IN-1 filing system that contains documents. There are two kinds of folders: personal and *system.* Personal folders are created and named by individual ALL-IN-1 users; system folders are created by ALL-IN-1 and cannot be deleted or renamed by the user. Some examples of system folders are *Inbox, Outbox*, and *Wastebasket.*

folder name

An optional *field* the user can include in a document's *header information* to specify a particular *folder.*

form

A *system* image that ALL-IN-1 displays to collect or show information the user needs to complete a *command.* This image is created with *FMS.*

ALL-IN-1 has several types of forms. See *menu form, entry form*, and *calendar form.*

form field

The area of a *form* that is used to *display data* or to accept and display data *input* by the user.

Form Set

A multipage form.

format

The way information is arranged in a *form*, *document*, or *file*. In a document or *message*, the *page* arrangement as determined by rulers, spacing, and page breaks.

Forms Development

The ALL-IN-1 *subsystem* that allows the user to *create* or *edit FMS* forms, to create FMS Form Language files, or to create *DATATRIEVE* definition files.

Forms Management System (FMS)

A set of *software* tools that simplify the development and maintenance of *application* programs involving formatted *input* and *output* to video terminals.

formula

A combination of operations in a logical computation in the *Desk Calculator application*.

Formula Functions

The mathematical functions in the *Desk Calculator*, such as ACOS, SIN, TAN, ATAN.

Formula Mode

In the *Desk Calculator*, the operating state that lets the user perform algebraic operations, such as finding the log (LOG) or square root (SQRT) of a number, or trigonometric functions, such as finding the cosine (COS) or arc sine (ASIN) of a triangle.

function

An ALL-IN-1 action *command*. Functions are analogous to *DCL* commands.

function key

Any key that causes the *system* to perform a function, such as clearing the *screen* or executing a *program*. Some function keys, such as DELETE, HYPH PUSH, and the arrow keys, have predefined actions.

Global Search-and-Replace

An editing step that locates every occurrence of a *word* or *phrase* defined by the user, deletes it, and in its place inserts new *text* also defined by the user. The contents of a *paste buffer* can be specified as the replacement text.

Gold key

The gold colored key, or PF1 key on the keypad, which has an effect only when pressed before another key. For instance, if the user presses the Gold key and then the I key, ALL-IN-1 displays the *Interrupt Menu* on the *screen*.

hard copy

Text printed on paper.

Hard Copy Mode

One of two modes in which ALL-IN-1 can operate. Instead of displaying forms, ALL-IN-1 displays prompts. (See also *Screen Mode*.)

hard return

A return or *carriage return* inserted by pressing the *RETURN* key.

hardware

The physical equipment in a computer *system*, for example, the central *processor*, terminals, printers, and *disk* drives.

header

Text that appears in the top *margin* of printed pages, such as a *document title*, *page* number, or date. Use *DIGITAL Standard Runoff* to *create* headers.

The control information prefixed in a *message text*, such as source or destination code or message type.

header information

Data that uniquely identifies a *document* or *message* in the ALL-IN-1 *File Cabinet*. Header information can include a *document title*, subject, author, *folder name, document number*, keywords, and creation date.

Help

Messages displayed on the *terminal screen* that provide information on how to use applications and other *system* services.

highlight

To *center*, underline, bold, or capitalize *text* in a *document*.

host

Another computer *system* that communicates with the ALL-IN-1 system.

Inbox

In the *Electronic Messaging* Subsystem, the *folder* where received messages are held until they are read.

Index

An ALL-IN-1 *option* that displays a list of all the documents stored in the user's *File Cabinet* or in a specified *folder.*

Information Management

The ALL-IN-1 *subsystem* that lets the user access the *Calculations, Communications,* and *Directories* subsystems.

Input

To enter information into ALL-IN-1, usually at a *form field.*

Information provided by the user that is read by a *program.*

Insert

To place new *text* in a *document.*

Installation Verification Procedure (IVP)

A script invoked after ALL-IN-1 has been installed or after the *system* has been modified.

Interactive

To communicate directly with the computer by typing commands at the *keyboard.*

Interrupt Menu

A *menu* that is invoked by pressing Gold I from anywhere in the ALL-IN-1 *system.*

invoke

To call features or menus to the *screen.*

To start an ALL-IN-1 *application.*

IVP

See *Installation Verification Procedure.*

justify

To *print* a *document* with a straight *right margin* so that the last *character* on each line falls in the same column position.

keyboard

The part of the *terminal* on which the user types instructions and *text.* The keyboard consists of the *main keyboard* and the *keypad.*

keypad

The smaller set of keys on the *keyboard* containing the editing keys.

keystroke

A single stroke of a key on the *keyboard*; pressing a single key, or combination of keys as in Gold L.

keyword

A *word* or part of a word used when displaying a selective index of documents.

laser printer

A type of *letter quality printer* that places characters on the paper by means of a laser instead of a print wheel.

left margin

The column position marking the start of each line.

letter quality printer (LQP)

Any printer that produces printed copy comparable in quality to that of a typewriter.

library

A single *file* that serves as an indexed collection of other files. Libraries are maintained by the *VMS* Librarian through the *DCL* LIBRARY command (with the exception of *FMS* form libraries, which are maintained through the FMS command).

ALL-IN-1 uses form libraries (file type .FLB) and help libraries (file type .HLB).

link

The connection between two computers; the connection between the *terminal* and the *host* computer.

List form

An ALL-IN-1 *form* from which the user chooses an entry for the current field.

local symbol

An ALL-IN-1 symbol that begins with a # and can be used to pass arguments and other data between ALL-IN-1 facilities. ALL-IN-1 only maintains local symbols for the duration of a session.

local symbol table

A high-speed, temporary symbol table which ALL-IN-1 maintains internally for the duration of an ALL-IN-1 session.

Logical Name

A user-specified name that is associated with a value, or equivalence name. Like DCL symbols, logicals are maintained in tables, but logical-name tables are maintained for a process, a group, or the whole system.

Logicals are frequently used in file specifications and are recognized by those ALL-IN-1 functions and qualifiers that take file specifications as parameters and arguments.

log in

To perform the necessary actions at a terminal that identify the user to the ALL-IN-1 *system.*

To perform a sequence of actions at a *terminal* that establishes a user's interactive *link* with the *operating system* and sets up *default* characteristics for the user's terminal session.

log out

To perform the necessary actions at a *terminal* that end the user's *interactive* communication with the *operating system.*

LQP

See *letter quality printer.*

mail

See *Electronic Messaging.*

mailbox

A record-oriented input/output *device* for communicating between processes. ALL-IN-1 creates three mailboxes, *DCLMAILBOX*, *OAMAILBOX*, and SYS$INPUT, to communicate between the main process and the *subprocess.* See also *Inbox* and *Outbox.*

main keyboard

The set of keys on the *keyboard* that looks like a typewriter keyboard.

Main Menu

The initial ALL-IN-1 *menu* from which the user may choose from among many *options.* The user's *default menu* when first entering ALL-IN-1.

margin

The space between the edge of the paper and the main body of the *text* on the *page.*

meeting

An activity the user schedules for a specific time and date that involves more than one user.

memory

A series of physical locations into which data or instructions can be placed in the form of binary words. Each location in memory can be addressed and its contents can be altered.

Memory Register

An area on the *Desk Calculator form* into which the user may place the results of computations for use with other computations.

menu

A displayed list of available *options* or choices associated with an ALL-IN-1 *subsystem.*

message

In the *Electronic Messaging* Subsystem, *text* that can be sent to or is received from other ALL-IN-1 users. A message is a type of *document* and is treated as such by the *File Cabinet.*

mode

One of two or more possible operating states within an ALL-IN-1 *application.* For example, the *Desk Calculator* can operate in either *Calculator Mode* or *Formula Mode.*

modem

A *communications device* that allows transmission of computer *data* over telephone lines.

module

A portion of a *program* or program library, such as a source module, object module, or image module.

Named Data

A *feature* of *FMS* that allows information to be stored "behind" a *form*. ALL-IN-1 uses Named Data to store the form .TYPE, form and field qualifiers, keypad definitions, *menu options*, and other information.

nesting

To include a *User Defined Process* inside of another.

network

A collection of individual computer systems linked together by *communications* lines allowing them to transmit *data* to each other.

NEW PAGE marker

A *character* that forces all *text* following it to the top of the next printed *page*; generally placed in a *document* at the start of a new chapter or section.

NEXT SELECTION key

The SEL (.) key on the *keypad* that lets the user *display* the name of the next *document* in a list of selected documents compiled using the *Index* (I) *option* on most of the ALL-IN-1 *subsystem* menus.

Nickname Management

The ALL-IN-1 *subsystem* that allows the user to *create* and maintain a list of nicknames for use as addressees in *Electronic Messaging* and *Time Management.*

node

An individual computer *system* in a *network* that can communicate with other computer systems in the same network.

numeric

Consisting of only numbers. See also *alphanumeric.*

OAMAILBOX

The *VMS mailbox* that ALL-IN-1 creates to *send* information from the *subprocess* to the main process. Used by command procedures and programs running in the subprocess.

on-line

Information that exists on the computer *system* and is accessible while the user is logged on.

operating system

The *software* that supports or complements the *hardware,* as by keeping track of the *input* and *output* of *data,* or executing programs or applications. (See also *VAX/VMS operating system.*)

operation

The performance of a specific process or activity, for example, sending *mail.*

options

The list of the user's possible choices appearing on a *menu* within ALL-IN-1.

Outbox

In *Electronic Messaging*, the *folder* where messages created by the user reside once they are sent.

output

The results produced by ALL-IN-1 in the form of a visual *display*, a *printout*, or a *file.*

overlay

The information that replaces a portion of a previous *display.*

overstrike

To superimpose a *character* on top of the current *character.* This erases the current *character.*

overwrite

To put *text* in a *document*, replacing whatever text was there before. When a user files a document at the end of an editing session, ALL-IN-1 overwrites the previous version of the document.

page

The part of a *document* that will be printed on one sheet of paper.

page break

To divide the *text* of a *document* into pages, either manually or automatically, using ALL-IN-1.

page end

Anything that determines where one *page* ends and another begins. (See *PAGE marker* and *NEW PAGE marker.*)

PAGE marker

A marker in a *document* that tells ALL-IN-1 to begin printing on the next *page.*

page size

The total number of single-spaced lines of *text* that can fit on a piece of paper.

paper copy

See *hard copy.*

paper mail

See *hard copy.*

paragraph

Text preceded or followed by two or more RETURNs or a paragraph marker.

paragraph indent marker

A marker on a *ruler* that determines at what column a *paragraph* will begin.

password

A string of characters that, in conjunction with other information such as the user's name, uniquely confirms the user's identity to the *system*. Must be *input* when the user logs in to the computer.

paste

To insert the contents of a *paste buffer* into the *document* being typed or edited by pressing PASTE. First, *text* must be *cut* into a paste buffer.

paste buffer

See *buffer*.

PDAF

See the *Personal Document Attributes File*.

permanent symbol

A global ALL-IN-1 symbol which must begin with a dollar sign ($) and may contain up to 30 characters. ALL-IN-1 limits permanent symbol names to the following characters: letters, numbers, a dollar sign ($), or an underscore (_).

permanent symbol table

An indexed data file that ALL-IN-1 creates as OAUSER:parsed-username.PST and saves from one session to the next.

Personal Document Attributes File (PDAF)

File OAUSER:DAF.DAT. Attributes of documents that are to be shared (such as mail messages) are stored in the PDAF when you first create a document.

phrase

A group of words shorter than a *sentence*. ALL-IN-1 recognizes a phrase if it is terminated by a punctuation mark or a line termination marker.

print

To direct ALL-IN-1 to produce a *hard copy* of a *document*.

print priority

A setting controlling the order in which documents waiting to *print* will be printed. (See also *priority*.)

print queue

The list of documents waiting to be printed.

print settings

Settings that control the manner in which a *document* is printed, such as the printer *margin* and the pitch.

printer-port

The connection on a *terminal* that allows a printer to be connected directly to the terminal.

printout

See *hard copy.*

priority

The order in which documents are printed or programs are processed. Documents with high priority will be processed first; documents with low priority will be processed last. (See also *print priority.*)

privilege

Values that determine the *system* resources you can access. In *VMS*, the SET PROCESS/PRIVILEGE=(PRIVLIST) command assigns privileges. In ALL-IN-1, privileges are based on the values of *User Profile* fields.

processor

The functional part of the computer *system* that reads, interprets, and executes *software* programs.

Profession Specific

The ALL-IN-1 *subsystem* that lets the user place personal or unique programs or applications on the Profession Specific *menu* and then access them from that menu.

program

The complete sequence of instructions and *data* necessary for the computer to perform a desired *operation.*

Program Development

The ALL-IN-1 *subsystem* that allows the user to access development tools. An interface for developing applications to integrate into or run independently of the ALL-IN-1 *system*.

Programming Functions

An ALL-IN-1 *subsystem* that allows the user to *create* and *edit application* programs.

prompt

A message ALL-IN-1 displays indicating the user must enter *text* or a *command*, or answer a question.

queue

See *print queue*.

Read

An ALL-IN-1 *option* that allows the user to view a *document*, *message*, or Time Management item, but not to make changes to it.

Read Receipt

A *mail message* sent by the ALL-IN-1 *system* to the originator of the message stating that the message has been read by the specified addressee.

Recognition

A *feature* that allows a user to request a list of all matching entries for a *field*. By *default*, Recognition is activated by keys Gold L and Gold S. It is available in fields that are qualified by /INVALID, /RECOGNITION, /VALID, and in key fields of *entry forms*.

record

A set of related data that a program treats as a unit.

Record Management Services (RMS)

A set of *operating system* procedures called by programs to process files and records within files.

Record Selection Expression (RSE)

An expression in which one or more data sets are searched using rules given in a *Boolean expression.* RSEs are used in FOR function calls.

reminder

An activity the user schedules to occur on a specific date but without a specific time assigned to it.

replace

To make ALL-IN-1 *delete* selected *text* in a *document* and *insert* the text in the *paste buffer* in its place. (See also *Global Search-and-Replace.*)

response time

The time ALL-IN-1 requires to perform an *operation.*

The elapsed time between the generation of the last *character* of a *message* at a *terminal* and the receipt of the first character of the reply.

restore

To return to a previous condition.

To copy *backup data* to recover the contents of a *file.*

RETURN

See *carriage return.*

right margin

The column position at which the last *character* of each line may appear.

RMS

See Record Management Services.

RSE

See Record Selection Expression.

ruler

The marker that defines the margins, tabs, indents, and vertical line spacing of all *text* following it. Any number of rulers may be placed in a *document.*

RUNOFF

See DIGITAL Standard Runoff.

screen

The *display* area of the *terminal.*

Screen Mode

One of two modes in which ALL-IN-1 can operate. Instead of displaying prompts, ALL-IN-1 displays forms. (See also *Hard Copy Mode.*)

scroll

To move forward or backward through a *document*, moving *text* up or down on the *screen* as though it were a scroll of paper; also, moving text left and right across screen windows.

SDAF

See the *Shared Document Attributes File.*

search

To make ALL-IN-1 find a particular *word* or *phrase* in a *document* and place the *cursor* at its start.

select

To choose a *menu* item by entering the appropriate *character* or characters. (See options.)

Selection List

A sequential *file* containing key values from an indexed file or other *data set.* Selection lists can be created by Search forms, FOR function calls, and the /LIST Select-form qualifier. They can be accessed through the OA$FLD_NEXT_LIST or NEXT_LIST functions.

Send

An ALL-IN-1 *option* that allows the user to transfer documents or messages to another user or *system.*

sentence

Text followed by a period, question mark, exclamation point, *RETURN*, or *PAGE marker.*

Setup

A *File Cabinet document* that is placed in the editor before you begin editing. Setups are usually MERGE templates that allow *symbol* substitution and other preprocessing.

SET-UP key

The key on the *main keyboard* that allows the user to change the characteristics of the *terminal.*

Shared Document Attributes File (SDAF)

File OA$DATA:DAF.DAT. Attributes of shared documents (such as sent mail messages) are stored in the SDAF.

software

The entire set of programs, procedures, and related documentation associated with a computer *system*; computer programs.

special symbol

An ALL-IN-1 symbol which contains a value that ALL-IN-1 predefines. Special symbols begin with a two- or three-character facility code such as DTR or OA, followed by a $.

special symbol table

A fixed set of symbols which are built into ALL-IN-1 Flow Control. The symbol set consists of read-only, write-only, and read-write symbols.

stored print settings

A set of *print* settings saved for later use.

stored ruler

Ruler settings saved for later use.

Subprocess

A process created and governed by ALL-IN-1. The ALL-IN-1 subprocess allows you to enter *DCL* level while running ALL-IN-1.

subsystem

An ALL-IN-1 *application* that can perform many functions; for example, *Desk Management* (DM) and *Electronic Messaging* (EM).

swap

To exchange or transpose the *character* at the *cursor* with the one to its right.

Symbol

A name that represents a value. In *DCL*, symbols are stored in local and global symbol tables. In ALL-IN-1, symbols can be stored in local, permanent, and special symbol tables; they can also represent values stored in fields on a *form* and data stored in files.

system

An entire ALL-IN-1 system, including the central *processor*, its *software*, terminals, printers, storage units, and other devices under its control.

system manager

The person responsible for the operation of an ALL-IN-1 *system*, the allocation of user accounts, and maintenance of the system files. Also responsible for the *VAX/VMS operating system*.

tab

To move the *cursor* a predetermined number of spaces across the line.

To move the cursor to the next field on an entry form.

tab stop

A *character* the user inserts into the *ruler*, which indicates where the *cursor* will move in the *text* when the TAB key is pressed.

table of contents

A list of the chapters and subheadings contained in a *document.*

Template

An *ASCII file* that contains MERGE-function directives and background text. Templates are prototype documents that are used in list-processing applications and anywhere boilerplate-type information is required.

terminal

A *device* for entering information to and receiving information from ALL-IN-1, consisting of a *keyboard* and either a video *screen* or a printer.

text

The words, characters, numbers, and information that constitute a *document* or other item the user creates.

TIME

An ALL-IN-1 *option* that displays the current date and time.

Time Management

The ALL-IN-1 *subsystem* that lets the user maintain a daily *calendar* of appointments, meetings, action items, and reminders, and maintain a personal *To-Do Lists.*

To-Do List

List of activities the user wants to complete but does not schedule a specific date.

top of document

The beginning of a *document.* Pressing Gold T, in the WPS-style editor, moves the user directly to the top of the document.

Training

The ALL-IN-1 *subsystem* that accesses the *CBI.* (See also *Computer Based Instruction.*)

TXL

A collection of text elements which consists of SCRIPT scripts, DO scripts, and MERGE boilerplates.

type

Enter *text* into a *document* or *message.*

UDP

See *User Defined Process.*

User Defined Process (UDP)

A set of steps an ALL-IN-1 user defines and stores for future use; may include commands, keystrokes, and *text.*

user name

The name a user types on a *terminal* to *log in* to the *system.*

User Profile

The ALL-IN-1 accounting *file.* Each ALL-IN-1 user is identified by an entry in the User Profile. These entries contain personal context information and *privilege* settings.

User Setup Parameters

The ALL-IN-1 *subsystem* that allows the user to modify that user's *User Profile*

Validation Expression

An expression against which the contents of *field* are compared. Validation expressions serve as arguments for validation qualifiers.

VAX

Virtual Address Extension; a memory-management *feature* of the computer of the same name. (See also *VAX/VMS operating system.*)

VAX/VMS operating system

The set of programs that manages the resources of a *VAX* computer to provide efficient computer operation.

verify

An ALL-IN-1 *facility* for checking documents, indexes, and task files for structural errors.

view

To make ALL-IN-1 *display text* in a way that shows characters that are normally invisible, such as carriage returns and word wraps.

VMS

Virtual Memory System; the *VAX operating system.* (See *VAX/VMS operating system.*)

VMS filename

The name of a *file* in the user's VAX/VMS directory.

Wastebasket

An ALL-IN-1 *folder* into which documents deleted from the user's ALL-IN-1 *File Cabinet* are placed.

word

To ALL-IN-1, a word is any group of visible characters separated by one of the following: a space, a *RETURN*, a *tab*, a *paragraph* marker, a *PAGE marker*, or a *NEW PAGE marker.*

Word and Document Processing

The ALL-IN-1 *subsystem* that lets the user *create*, *display*, edit, and/or *print* documents. The user also can list and search for documents in the *File Cabinet.*

word processing system

A *software system* that processes *text* performing such functions as text editing, paragraphing, paging, left and right justification, rearrangement of lines, and printing. Also called a word processor.

word wrap

The automatic dropping to the next line of any *word* a user types if it extends past the *right margin* of the *text.*

workstation

The user's ALL-IN-1 *terminal* and *keyboard* connected to a *VAX* computer *system*; also may include a printer.

WPS editor

An editor designed for editing documents that were created on a *word processing system.*

Index

This index references pages in all volumes: Volume 1 comprises the Preface through Chapter 5; Volume 2 comprises Chapter 6; Volume 3 comprises Chapter 7 through the Appendixes and the Glossary.

" and '
  string delimiters
    and /COMPUTE, 4-7
    and /GET form qualifier, 3-16
    and /HARD, 3-16, 4-18
    and /LOG, 3-19
    and /POST_FUNCTION, 3-21, 4-24
    and /PRE_FUNCTION, 3-21, 4-25
    and /PROMPT, 4-26
    and /SHOW, 4-45
    and COMMAND function parameters, 6-32
    and CXP SEND, 12-8
    and CXP WAIT, 12-9
    and FOR function RSEs, 6-88
    and GET, 6-107
    and OA$TABLE, C-8
    and QUEUE_BATCH qualifiers, 6-280
    and QUEUE_PRINT qualifiers, 6-285
    in symbols, 8-2
    in subprocess *See also* ', 7-6
",'
  string delimiters
    in DSRs, 8-6
#
  local-symbol prefix, 8-4
$
  and DCL function, 6-51
  permanent-symbol prefix, 8-3

&
MERGE directive prefix, 6-138
'
See also " and '
symbol-substitution character
in subprocess, 7-6
', " and string delimiters
and OA$TABLE, C-8
-
continuation character
and COMPUTE, 6-41
in WRITEs
to OAMAILBOX, 7-7
.
as field identifier in
FOR-function RSEs, 6-89
FORM-function substitution
character, 6-95
substitution character, 3-4
/
function-qualifier delimiter, 6-3
:
as a symbol-substring delimiter,
8-10
as an alternate-key delimiter
in DSRs, 8-6
Mail distribution list suffix,
11-3
<
function-call prefix, 3-32, 6-2
in a validation expression,
4-20, 4-30, 4-49, 4-54
pipeline source prefix, 6-9
<->
continuation characters
for MERGE template
directives, 6-139
=
buffer identifier in default
editor, 10-5
Help cross-reference prefix,
14-16
@
iterative symbol substitution
prefix, 8-12
Mail distribution list prefix,
11-3
Remote-node mail prefix, 11-4
\
function-call delimiter
function list, 3-21
in subprocess, 7-7
in user-defined
functions, 6-11
\\
function call delimiter
in .IF expressions, 15-18
function-call delimiter
in FOR-function calls, 6-87
_
SPECIAL mail destination
prefix, 11-4

A
ACMSTASK function, 6-13 to
6-15
Action Items
checking
with AI_CHECK, 6-17 to
6-18
class, 3-42
entry form AIE
field information, 3-40
item, 3-42
priority, 3-42
schedule date, 6-17
status, 6-17
storage of, 3-40
type, 3-42, 6-17
Additional options
and Gold A, 4-56
ADE function, 6-16
AI_CHECK function, 6-17 to
6-18
ALL-IN-1
and the VAX Information
Architecture, 1-4
customizing
suggested prerequisites, xliii
exercises, A-1 to A-22

facilities
primary, 1-7
flow control, 1-4
organization, 1-6
overview, 1-1 to 1-7
performance
See Performance
subsystems
overview, 1-6
symbols, 8-1 to 8-36
APPEND function, 6-19 to 6-20
Application
debugging an
with Gold (, 6-34
defined, 1-6
design guidelines, 16-18
"standard", 16-19
subsystem versus, 1-6
APR
conventions, 7
organization of, 2
overview, 1 to 8
related documents, 4, 5 to 7
using the, 3
Architecture, VAX Information
overview, 1-3
Argument form, 3-56 to 3-57
Named Data, 3-56
qualifiers, 3-57
.TYPE ARG, 3-5
ATTACH function, 6-21 to 6-22
Attributes, document, 15
Attributes, FMS field
used to define a field, 4-2
/AUTO field qualifier, 4-5
and /RECOGNITION, 4-28
and /RSE_RECOG, 4-31
Autorecognition
with /AUTO, 4-5

B

BACKSPACE
and OA$FLD_PREV, 4-56
Batch jobs
See QUEUE_BATCH
/BEGIN form qualifier, 3-10, 3-27, 3-47
and /FIELDS, 3-49
in menus, 3-11
/BLANK field qualifier, 4-5 to 4-6
Boolean Expression, 4
Boolean expression, 6-86
in CABINET SCAN calls, 9-44
in FOR-function calls, 6-85
in search-form RSE, 3-60, 3-63
Boolean operators, 4
in .SFILE directive, 3-66
in FOR-function calls, 6-85
in search form RSE, 3-64
Brace character, left
used as script mnemonic
delimiter, 15-3
Brace character, right
used as script mnemonic
delimiter, 15-3
BYE function, 6-23 to 6-24

C

CAB$
File Cabinet DSR, 9-16
CAB$ATTACH, 9-17
CAB$ATTRIBUTES, 9-17
in a search form, 3-63
in a select form, 3-71
CABINET function, 6-25, 9-20 to 9-47
subfunctions
See also CABINET HELP
ADD_ATTRIBUTE, 9-21
ATTACH_DOCUMENT, 9-21
BEGIN, 9-22
CHANGE_ATTRIBUTE, 9-22
CLEAR_CURDOC, 9-23
CLOSE, 9-23
CONVERT, 9-23
COPY, 9-24
CREATE, 9-25

CABINET function,
CROSSFILE, 9-26
CURRENT, 9-27
DECREMENT_COUNT, 9-27
DELETE_ATTRIBUTE, 9-27
DELETE_DOCUMENT, 9-28
DELETE_OR_REFILE, 9-28
DETACH_DOCUMENT, 9-29
EDIT_DOCUMENT, 9-30
END, 9-30
EXCHANGE, 9-30
FILE_ATTACHMENT, 9-30
FORCE, 9-31
GET_PENDING, 9-31
HELP, 9-32
JANITOR, 9-33
NEXT_DOCUMENT, 9-33
NEXT_FOLDER, 9-35
NEXT_NUMBER, 9-37
NEXT_SCAN, 9-38
NEXT_TITLE, 9-40
OPEN, 9-42
POP, 9-42
PUSH, 9-42
PUT_PENDING, 9-42
REFILE_DOCUMENT, 9-43
REORG_DOCDB, 9-44
RESET_DELETE, 9-44
See also CABINET RESET_DELETE
SCAN, 9-44
SELECT, 9-45
SHARE_DOCUMENT, 9-9, 9-46
STOP_SCAN, 9-47
summary, 9-13
UNSHARE_DOCUMENT, 9-47
Calculation form, 3-57 to 3-59
behavior, 3-57
Named Data, 3-57
qualifiers, 3-58
.TYPE CALC, 3-6
.TYPE CLEAR, 3-6
.TYPE DTR, 3-6
CALENDAR function, 6-26
current date/time, 13-2
data sets, 13-1
used by, 13-30
files used by, 13-3
subfunctions, 13-3 to 13-27
CALENDAR ADD_ATTENDEE, 13-3
CALENDAR CANCEL APPOINTMENT, 13-4
CALENDAR CANCEL MEETING, 13-4
CALENDAR CLEAR ATTENDEES, 13-5
CALENDAR CONVERT ACTION_ITEMS, 13-5
CALENDAR CONVERT APPOINTMENTS, 13-5
CALENDAR CONVERT MEETINGS, 13-6
CALENDAR DISPLAY ANSWER, 13-6
CALENDAR DISPLAY APPOINTMENT, 13-7
CALENDAR DISPLAY GRAPH, 13-8
CALENDAR DISPLAY MEETING, 13-9
CALENDAR DISPLAY MONTH, 13-9
CALENDAR DISPLAY NEXT APPOINTMENT, 13-10
CALENDAR DISPLAY NEXT MEETING, 13-11
CALENDAR DISPLAY NEXT REPLY, 13-11
CALENDAR DISPLAY REPLY, 13-12
CALENDAR INITIALIZE FILES, 13-13
CALENDAR INITIALIZE MONTH, 13-13

CALENDAR MAIL_TO
_ATTENDEES, 13-14
CALENDAR
MAKE_ATTENDEE_
SYMBOLS, 13-14
CALENDAR
MAKE_MEETING_
SYMBOLS, 13-15
CALENDAR PRINT, 13-16
CALENDAR PRINT
ALL, 13-16
CALENDAR PRINT
CUSTOM, 13-17
CALENDAR PRINT
DAY_REMIND, 13-17
CALENDAR PRINT
DAY_SCHEDULE, 13-18
CALENDAR PRINT
MONTH, 13-18
CALENDAR PRINT
TODO, 13-19
CALENDAR PRINT
TWOCAL, 13-19
CALENDAR PRINT
WEEK_REMINDERS,
13-20
CALENDAR PRINT
WEEK_SCHEDULE,
13-20
CALENDAR PURGE, 13-20.1
CALENDAR
REMOVE_ATTENDEES,
13-20.1
CALENDAR
REPLY_TO_MEETING,
13-21
CALENDAR RESCHEDULE
APPOINTMENT, 13-21
CALENDAR RESCHEDULE
MEETING, 13-22
CALENDAR SCAN, 13-22
CALENDAR SCHEDULE
APPOINTMENT, 13-23
CALENDAR SCHEDULE
MEETING, 13-23
CALENDAR SET
ALTERNATE_USER,
13-24
CALENDAR SET
DATE, 13-25
CALENDAR SET
NETWORK OFF, 13-26
CALENDAR SET
NETWORK ON, 13-26
CALENDAR VERSION,
13-27
/CAPTIVE menu qualifier, 3-26
CBI
exit keys
Gold X, CTRL/C, 4-59, 15-4
Character, left Brace
used as script mnemonic
delimiter, 15-3
Character, right Brace
used as script mnemonic
delimiter, 15-3
CHECK_DISKQUOTA function,
6-26.1 to 6-26.2
/CHOICE menu qualifier, 3-25
/CLEAR field qualifier, 4-6
/CLEAR form qualifier, 3-11
and /CHOICE, 3-11
CLEAR function, 3-21, 6-27 to
6-28
versus /OVERLAY, 6-28
CLOSE_PRIOR function, 6-29 to
6-30
in function lists, 6-7
COMMAND function, 6-31 to 6-36,
7-19
and mailboxes, 7-2
debugging tools used with, 6-34
passing parameters with, 7-16
simulating search strategy of
with OA$LIB:FINDCOM,
6-35
versus QUEUE_BATCH, 6-282
Command procedure
debugging
with Gold (, OA$LIB:V.COM,
6-34

Command procedures versus
scripts performance in
applications, 16-22
Common Data Dictionary (CDD), 5
Communications Control, 12-2 to
12-13
See also CXP function
and the CXP function, 6-47
Control Documents, 12-2
customizing, 12-12
KEYPAD function, 6-117
Terminal Emulation Mode, 12-2
/COMPUTE field qualifier, 4-7
COMPUTE function, 6-37 to 6-41
from the subprocess, 7-10, 7-12
Compute key
Gold #, 4-58
Control Documents, 12-2
CONTROL_C function, 6-42
Converting to Version 2.X
form libraries, 16-9
/COPY field qualifier, 3-79, 4-8 to
4-10
order of processing, 4-10
COPY function, 6-43 to 6-44
as a pipeline source, 6-10
in Pipeline Mode, 6-44
Copy Mode, hard, B-1 to B-5
See also MODE function
providing information while in,
4-18
CREATE function, 3-38, 6-45 to
6-46
Customizing ALL-IN-1
performance considerations
See Performance
with user-defined functions,
6-10 to 6-11
Customizing documentation
for HELP, 16-24
for the user, 16-23
Customizing forms, 16-8
CXP function, 6-47, 12-2 to 12-13
See also External
Communications,
Communications Control
subsystem
subfunctions, 12-5
AT, 12-5
CONNECT, 12-5
CONNECT/DEVICE, 12-6
DISCONNECT, 12-6
DISPLAY, 12-6
PAUSE, 12-6
PROCESS, 12-7
RECORD, 12-7
restrictions on calling, 12-9
SEND, 12-8
SEND/DOCUMENT, 12-8
SET, 12-9
STOP RECORDING, 12-9
WAIT, 12-9

D

/DATA entry form qualifier
and /KEY, 3-48
Data file keys
ALL-IN-1 versus RMS, 3-39
specified with FDL, 3-39
Data files
accessing
through alternate keys, 8-6
through DSRs, 8-4 to 8-9
with .FILE, 3-44
with /DATA, 3-44
adding unique key values to
with /INVALID, 4-21
converting
with FOR and WRITE, 6-91
copying
with COPY, 6-43 to 6-44
creating, 3-38, 16-4
with CREATE, 6-45 to 6-46
with OA$INI_LOGICALS,
6-204
deleting, 6-56 to 6-57
modifying
with WRITE, 6-328 to 6-329

updating existing records
with /VALID, 4-49
upgrading
with OA$CNV_CONVERT, 6-161 to 6-162
with OA$CNV_MERGE, 6-165 to 6-166
Data set, 11
See also DSR
text
See Text data sets
Data sets, C-1 to C-15
caching, 6-69, 6-169
CALENDAR, 13-1
data, C-1
See also DSR
form, C-2
in a validation expression, 4-20, 4-29, 4-48, 4-51
in FOR function RSEs, 6-84
permanently open, 6-70
special
See DSR
text, C-2
Data validation
See Validation
See also DTR function
and DTR function, 6-66
and DTR$ special symbols, 8-3
creating record definitions, 16-5
getting information from, 6-66
/DATE form qualifier, 3-12
Date/time formats, 6-49, 8-13
DATE_CONVERT function, 6-48 to 6-50
from the subprocess, 7-10
Dates and times
formatting with DATE_CONVERT, 6-48 to 6-50
DCL
noninteractive commands
with OA$DCL, 6-52
DCL form
.TYPE DCL, 3-6
DCL function, 6-51 to 6-53, 7-19, 16-15
and Gold $, 4-58
and mailboxes, 7-2
called from the editor, 16-16
from the default editor, 10-17
DCL key
Gold $, 4-58
DCL symbols
common, 7-12 to 7-15
and function lists, 7-15
RESULT, 7-12
TERMINATOR, 7-13
creating
with /COPY, 4-8, 7-9
with /PUT_SAVE, 4-27, 7-9
with COMPUTE, 6-38, 7-10
with DATE_CONVERT, 6-48, 7-10
with functions, 7-10 to 7-11
with GET, 6-102, 6-103, 7-11
with MAKE_FILE_NAME, 6-130, 7-11
with NEXT_LIST, 6-155
with NEXT_LIST, OA$FLD_NEXT_LIST, 7-11
with PARSE_USER, 7-11
with SAVE, 6-293, 7-11
verifying assignment of, 16-15
DCLMAILBOX
and OAMAILBOX, 7-4
mailbox to subprocess, 7-4
saving data that comes through
with RESULT, 7-12
with TERMINATOR, 7-13
Debugging
applications
with Gold ((SET VERIFY), 6-34
command procedures
with Gold (, OA$LIB:V.COM, 6-34
scripts
with Gold ((.TRACE ON), 6-34

Debugging applications
with Gold (, 4-62
Debugging features, 16-15
/DECIMAL field qualifier, 4-10 to 4-11
versus DECIMAL function, 4-11
DECIMAL function, 4-11, 6-54 to 6-55
Default menu
form DEFAULT, 3-32, 4-56
keypad definitions in, 3-32
setting
with SET_MENU, 6-300 to 6-301
Defining/redefining the keyboard, 4-63 to 4-65
DELETE_FILE function, 6-56 to 6-57
Desk Calculator form, 3-58 to 3-59, 4-60
formula mode, 3-59
mathematical functions, 3-59
setting decimal places, 3-59
.TYPE DESK, 3-6
using memory registers, 3-59
DESTINATION function, 6-58, 10-10
and the Editor menu, 6-75
DIGITAL Command Language
See DCL
Directives, keyboard
recognized at prompts, 6-269
/DISPLAY form qualifier, 3-12
DISPLAY function, 6-59 to 6-62
Displaying information
through a scrolled region, 4-33
with /SHOW, 4-45
Dispositions, form
terminator values, 4-69
DO function, 6-63 to 6-64
See also SCRIPT
DO key
RETURN or Gold F, 4-58
/DOC field qualifier, 4-11 to 4-14
and MAKE_FILE_NAME, 6-133
Document attributes, 15
Document, File Cabinet
See also File Cabinet, Text files
attachments, 9-3, 9-21
attributes, 9-3, 9-8, 9-9, 9-13
See also PDAF, SDAF
adding, 9-21
changing, 9-22
deleting, 9-27
compound, 9-3
detaching from, 9-29
filing attachments, 9-30
copying, 9-24
See also Text files, copying
creating, 9-25
cross-filing, 9-26
current, 9-27
See also OA$CURDOC
deleting, 9-28, 9-33
See also File Cabinet Wastebasket and Text files, deleting
with File Cabinet janitor, 9-4
editing, 9-30
See also Text files, editing
next
by folder, 9-35
by number, 9-37
by title, 9-40
in a scan, 9-38
in folder, 9-33
number, 9-2
ordering, 9-6
refiling, 9-43
removing from shared area, 9-47
selecting, 9-45
sharing, 9-3, 9-46
Documentation, customizing
for HELP, 16-24
for the user, 16-23

Down Arrow
and OA$FLD_DOWN, 4-58
calls OA$SCL_DOWN in scrolled region, 4-34
DSR, 8-4 to 8-9
alternate keys with, 8-6
and /DATA, 3-48
combining and nesting, 8-6
File Cabinet, 9-16
See also CAB$
in a validation expression, 4-51
with /INVALID, 4-20
with /RECOGNITION, 4-29
with /VALID, 4-48
obtaining data with, 8-8
scrolling
SCROLL$FUNCTION, 4-36 to 4-42
special, C-2
OA$DIR, C-3
special field names
%ADDRESS, 8-7
%COUNT, 8-7
%KEY, 8-8
%NEXT, 8-8
%WHOLE, 8-8
syntax, 8-4
validation
OA$DATE, C-2
OA$TABLE, C-8
OA$YESNO, C-9
OA$YN, C-9
DTR function, 6-65 to 6-68
DUMP_CACHE function, 6-69 to 6-70, 16-11
and OA$FLO_CLOSE_LIB, 6-194
and OA$FLO_OPEN_LIB, 6-196

E

Edit form
.TYPE EDIT, 3-6
EDIT function, 6-71 to 6-73
Editing (default editor), 10-4 to 10-19
See also EDIT function
Change Mode, 10-4
commands
nokeypad commands
switching to Line Mode from, 10-18
creating new files from editor, 10-17
edit buffers, 10-5
including text into document, 10-16 to 10-17
initialization file, 10-8
initializing, 10-4
key numbers, 10-12
Line Mode, 10-4
commands, 10-4
APPENDSR, 10-15
BUFFER, 10-5
CHANGE, 10-8, 10-9, 10-19
CLEAR, 10-9
COPY, 10-15
CUTSR, 10-14
DEFINE KEY, 10-6
DEFINE MACRO, 10-5
FIND, 10-9
INCLUDE, 10-16
INSERT, 10-9
PASTE, 10-14
RESET, 10-14
SEL, 10-14
SHOW KEY, 10-11
WRITE, 10-17
macros, 10-5
invoking with key sequences, 10-8
moving text within a document, 10-14 to 10-16
moving within a document, 10-13
nokeypad commands, 10-5
ADV, 10-13
and DEFINE KEY, 10-6
BACK, 10-13

Editing (default editor)
nokeypad commands (Cont.)
BR, 10-14
C, 10-13
DEFK, 10-6
ER, 10-13
EXIT, 10-19
EXTEND (EXT), 10-8
L, EL, 10-13, 10-14
V, 10-13, 10-14
W, 10-13
XLATE, 10-10
and DESTINATION, 10-10
Editing (WPS-PLUS)
converting files, 10-21
Editor menu
and EDT function, 6-74 to 6-75
and function-call output, 6-75
EDT function, 6-74 to 6-75
Electronic Messaging, 11-1 to 11-50
See also MAIL function
alternative MTS, 11-5
current message, 11-1
delivery mechanism, 11-3
distribution lists, 11-3
SUBSCRIBERS:, 11-4
header and text, 11-2
logicals
OA$MAIL_NUM_ADDRESSES, 11-9
mail destinations, 11-4
menu interface, 11-2
nicknames, 11-3
special addressees
and SPECIAL.CMU, 11-6
and SPECIAL.COM, 11-6
@dest @node, 11-4
dist_list:, 11-4
I or ME, 11-3
PAPER MAIL, 11-3
_special-addressee, 11-4
special folders, 11-2
unrecognized destinations, 11-6
Entry form, 3-37 to 3-56
and data sets, 4-51
dynamically redefining, 3-78
key field recognition
and /SHOW, 4-46
Named Data, 3-37
processing, 3-54 to 3-56
required features, 3-37
.TYPE ENTRY, 3-6
using /STORE with, 4-46
ERROR function, 6-76 to 6-77
in function lists, 6-6
Errors
logging
with LOG, 6-124 to 6-125
trapping in function lists with ERROR, 6-6, 6-76 to 6-77
EXIT function, 6-78 to 6-79
EXIT SCREEN
and the /POST_FUNCTION field qualifier, 4-24
and the prior form, 3-75
from the editor menu, 6-315
keypad 0
and OA$FLD_EXIT, 4-59
trapping with IFEXIT, 6-6
EXIT_SCREEN, 3-20
Expression, boolean
in search-form RSE, 3-63
Expression, record selection
See RSE
External Communications, 12-1 to 12-13
See also CXP function, Multinode features, Communications Control subsystem

F

Facility
defined, 1-6
FDL file specification
in entry form Named Data, 3-39
FHELP function, 6-80

Field
as a symbol, 8-2
assigning a permanent value to
with /BLANK, 4-5
assigning values dynamically
with PUT_FIELD, 6-277 to 6-278
attributes
assigned with FMS, 4-2
FMS versus ALL-IN-1, 4-3
clearing
with /CLEAR, 4-6
defined, 4-1
defining
with field qualifiers, 4-2
with FMS field attributes, 4-2
display-only, 3-13
getting values from
with /PUT_SAVE, 4-27
in a DSR, 8-4
layout
viewing with Gold V, 4-63
moving from one to another, 4-66
postprocessing from
with /POST_FUNCTION, 4-24
preparing for mathematical output
with /DECIMAL, 4-10
preprocessing from
with /PRE_FUNCTION, 4-25
prompting through
with /PROMPT, 4-26
putting file names into
with /DOC, 4-11
putting values into
with /COPY, 4-8
with /GET_SAVE, 4-16
suppressing input
with /BLANK, 4-5
validating contents of
with /FILE_CHECK, 4-14
validation, 4-51 to 4-56
See also Validation
ensuring a unique value
with /INVALID, 4-19
/VALID, 4-47
viewing valid values for
with /RECOGNITION, 4-28
FIELD function, 4-2, 6-81 to 6-82
and /FIELDS form qualifier, 4-2, 6-81
and OA$FLD_nnn functions, 4-67 Field processing, 4-1 to 4-70
summary, 4-69 to 4-70
Field qualifiers
/AUTO, 4-5
/BLANK, 4-5 to 4-6
/CLEAR, 4-6
/COMPUTE, 4-7
/COPY, 3-79, 4-8 to 4-10
/DECIMAL, 4-10 to 4-11
/DOC, 4-11 to 4-14
/FILE_CHECK, 4-14 to 4-16
/GET_SAVE, 4-16 to 4-18
/HARD, 4-18 to 4-19
/INVALID, 4-19 to 4-22
/LIST, 4-22 to 4-23
/OPTIONAL, 4-23 to 4-24
/POST_FUNCTION, 4-24 to 4-25
/PRE_FUNCTION, 4-25 to 4-26
/PROMPT, 4-26 to 4-27
/PUT_SAVE, 4-27 to 4-28
/RECOGNITION, 4-28 to 4-30
/RSE_INVALID, 4-30 to 4-32
/RSE_RECOG, 4-30 to 4-32
/RSE_VALID, 4-30 to 4-32
/SCROLL, 4-33 to 4-44
/SHOW, 4-45 to 4-46
/STORE, 4-46
types of
in-field, 4-4
postprocessing, 4-4
preprocessing, 4-4
validation, 4-4

Field qualifiers (Cont.)
/UNIQUE, 4-47
/USE_FORM, 4-47
used to define a field, 4-2
/VALID, 4-47 to 4-50
/FIELDS form qualifier, 3-13, 3-34
and /BEGIN, 3-13, 3-49
and /GET, 3-13
and /KEY, 3-49
and FIELD function, 6-81
in FORM function calls, 3-9
in menus, 3-26
File Cabinet, 9-1 to 9-47
accessing data, 9-15 to 9-18
See also Special symbols and CAB$ data set
as a data set, 9-15
attachments, 9-3
See also Document attachments
converting, 9-23
Document, 9-2
See also Document, Text files attached
See Document attachments compound
See Document, compound
number, 9-2
sharing, 9-3
folder, 9-3
janitor, 9-4
organization, 9-4 to 9-8, 9-13
See also individual file names
personal area, 9-4
reorganizing, 9-44
searching the
with search forms, 3-63
with select forms, 3-71
shared area, 9-19
switching
with NEWDIR, 6-150
using the, 9-13 to 9-19
Wastebasket
and CABINET JANITOR, 9-33
File Cabinet special symbols
See Special symbols, 9-18
File Definition Language
See FDL
.FILE directive
in entry form Named Data, 3-38
File names
creating
with /DOC, 4-11
/FILE_CHECK field qualifier, 4-4, 4-14 to 4-16
Files
See Text files, Data files
copying with COPY, 6-43 to 6-44
deleting
with DELETE_FILE, 6-56 to 6-57
managing
with the File Cabinet, 9-1 to 9-47
naming
with MAKE_FILE_NAME, 6-130 to 6-133
processing, 10-32
purging
with PURGE_FILE, 6-273 to 6-276
Flow control, 2-1 to 2-6
ALL-IN-1, 1-4
between ALL-IN-1 and its subprocess, 7-3
framework, 2-4 to 2-6
FMS field attributes
used to define a field, 4-2
FMS field highlights
performance in applications, 16-23
FMS forms
ALL-IN-1 use of, 3-1
FOR function, 6-83 to 6-91
and /RSE_nnn field qualifiers, 4-32
and validation qualifiers, 6-90

and WRITE
converting files, 6-91
called, 4-31
data sets in, 6-84
DO subfunctions
in select forms, 3-72
in search forms, 6-90
in select forms, 3-69, 6-90
RSEs in, 6-83
selecting unique records
with, 6-84
FORCE function, 6-92 to 6-93
in function lists, 6-7, 6-92
Form
caching, 6-69, 6-169
called directly from menu
choice field, 3-31
creating, 16-4
customizing, 16-8
displaying
with SHOW_FORM, 6-302
dispositions
and OA$FORM_DISPOSE,
4-66
DONE, 4-68
EXIT, 4-68
STAY, 4-68
summary of, 4-68
dynamically redefining, 3-8
maintaining, 16-2 to 16-12
memory-resident, 16-10
modifying, 16-5
multipage
See Form set
nonstandard, 3-80
redefining
with FORM function calls,
3-75 to 3-78
saving image of
with PRINT_SCREEN,
6-265
segmented
See Form set
testing, 16-11
Form (Cont.)
.TYPE
ARG, 3-5
CALC, 3-6
CLEAR, 3-6
DCL (special), 3-6
DESK (special), 3-6
DTR, 3-6
EDIT (special), 3-6
ENTRY, 3-6
HELP (special), 3-6
in FORM function calls, 3-76
LIST (special), 3-6
MENU, 3-6
Named Data entry, 3-5
SEARCH, 3-6
SELECT, 3-6
VIEW, 3-6
Form and function
in ALL-IN-1, 1-5
in the VAX Information
Architecture, 1-3
Form dispositions
terminator values, 4-69
FORM function, 3-3, 6-94 to 6-96
and OA$FLD_nnn
functions, 4-67
form .TYPE entry in calls to,
3-76
in function lists, 6-7
redefining Named Data with,
3-75
substitution character with, 3-4
Form Libraries, 3-2, 16-9
closing
with
OA$FLO_CLOSE_LIB,
6-193 to 6-194
interactively updating, 16-17
opening
with
OA$FLO_OPEN_LIB,
6-195 to 6-196
opening/reopening
with NEWLIB, 6-152 to
6-154

Form Libraries (Cont.)
search order, 3-3
searching, 16-9
status of
and
OA$FBT_LIST_LIBTREE, 6-175
and OA$FBT_LIST_TREE, 6-177
Form processing, 3-1 to 3-80
See also individual form
.TYPES
Form qualifiers, 3-8
in function calls, 3-8
common, 3-10 to 3-24
/BEGIN, 3-10
/CLEAR, 3-11
invalid with entry forms, 3-11
/DATE, 3-12
/DISPLAY, 3-12
/FIELDS, 3-13
/GET, 3-14
/HARD, 3-16
/HELP, 3-17
/HIGHLIGHT, 3-17
/KEYPAD, 3-18
/LOG, 3-19
/MAIL, 3-19
/NEXT, 3-19
/OVERLAY, 3-20
/POST_FUNCTION, 3-21
/PRE_FUNCTION, 3-21
/RESET, 3-22
/SUPER, 3-23
/TITLE, 3-23
/USER, 3-24
entry form
/BEGIN, 3-47
/DATA, 3-48
/FIELDS, 3-49
/KEY, 3-50
/MODE (required), 3-44
/NOWRITE, 3-52
/ONCE, 3-52
/ONE_ENTRY, 3-53
Form qualifiers (Cont.)
/SAVE_START, 3-46, 3-53
in function calls
rules for processing, 3-8
menu
/CAPTIVE, 3-26
/CHOICE (required), 3-25
/FIELDS, 3-26
/MORE, 3-27
/ONCE, 3-28
select form
/LIST, 3-69
/STYLE, 3-70
Form set, 3-78 to 3-80, 20
duplication of field names in, 3-79
entering first field in
with Gold T, 4-62
entering last field in
with Gold B, 4-57
entering next page in
with Gold TAB, NEXT SCREEN, 3-79, 4-60
entering previous page in
with Gold BACKSPACE, PREV SCREEN, 4-61
Gold key functions for, 3-80
restrictions on creating, 3-79
.TYPE directive in, 3-78
.FORM_SET directive
See also Form set
used to create a form set, 3-78
FORMAT entry form, 10-29
FORMAT function, 6-97 to 6-100
as a pipeline source, 6-10
Formatting files
See Text files, formatting
Forms, customizing, 16-8
Forms, precompiling
with the PFC option, 16-11
Function lists
creating loops in
with REPEAT, 6-7
ERROR in, 6-6
forcing displays in
with FORCE, 6-7

from the subprocess, 6-8, 7-7
and common DCL symbols, 7-15
IFEXIT in, 6-6
trapping error conditions in, 6-6
with /POST_FUNCTION qualifiers, 3-21, 4-24
with /PRE_FUNCTION qualifiers, 3-22, 4-25
Functions
See also individual functions
ACMSTASK, 6-13 to 6-15
ADE, 6-16
AI_CHECK, 6-17 to 6-18
APPEND, 6-19 to 6-20
ATTACH, 6-21 to 6-22
BYE, 6-23 to 6-24
called from validation expression, 4-20, 4-30, 4-49, 4-54
called through keyboard directives, 4-63
calling, 6-1
CHECK_DISKQUOTA, 6-26.1 to 6-26.2
CLEAR, 6-27 to 6-28
CLOSE_PRIOR, 6-29 to 6-30
COMMAND, 6-31 to 6-36
COMPUTE, 6-37 to 6-41
CONTROL_C, 6-42
COPY, 6-43 to 6-44
CREATE, 6-45 to 6-46
CXP, 6-47
DATE_CONVERT, 6-48 to 6-50
DCL, 6-51 to 6-53
DECIMAL, 6-54 to 6-55
DELETE_FILE, 6-56 to 6-57
DESTINATION, 6-58
DISPLAY, 6-59 to 6-62
DO, 6-63 to 6-64
DTR, 6-65 to 6-68
DUMP_CACHE, 6-69 to 6-70
EDIT, 6-71 to 6-73
Functions (Cont.)
EDT, 6-74 to 6-75
ERROR, 6-76 to 6-77
EXIT, 6-78 to 6-79
FHELP, 6-80
FIELD, 6-81 to 6-82
FOR, 6-83 to 6-91
FORCE, 6-92 to 6-93
FORM, 6-94 to 6-96
FORMAT, 6-97 to 6-100
general syntax, 6-2
GET, 6-101 to 6-107
HELP, 6-108 to 6-111
IFEXIT, 6-112 to 6-113
INCLUDE, 6-114 to 6-115
interactive
and terminator sequences, 7-13
internal terminator, 4-66
KEYPAD, 6-116 to 6-117
LIGHTS, 6-118 to 6-119
LIST, 6-120 to 6-123
listing the
with FHELP, 6-80
LOG, 6-124 to 6-125
LOGICAL, 6-126 to 6-128
MAKE_FILE_NAME, 6-130 to 6-133
MERGE, 6-134 to 6-140
MERGE_LINE, 6-141 to 6-142
MERGE_LIST, 6-143 to 6-146
MODE, 6-147 to 6-149
nesting
See FOR, MERGE, SCRIPT
NEWDIR, 6-150 to 6-151
NEWLIB, 6-152 to 6-154
NEXT_LIST, 6-155 to 6-156
OA$CLI_GET_SYMBOL, 6-157 to 6-158
OA$CLI_GET_VALUE, 6-159 to 6-160
OA$CNV_CONVERT, 6-161 to 6-162
OA$CNV_DIST_LIST, 6-163 to 6-164

Functions (Cont.)
OA$CNV_FROM_SCROLL, 6-167, 6-224 to 6-227
OA$CNV_MERGE, 6-165 to 6-166
OA$CNV_TO_SCROLL, 6-168, 6-224 to 6-227
OA$DSA_CACHE_STATUS, 6-169 to 6-170
OA$ENT_SET_xxx, 6-171 to 6-172
OA$FBT_INSTALL_LIBRARY, 6-173 to 6-174
OA$FBT_LIST_LIBTREE, 6-175 to 6-176
OA$FBT_LIST_TREE, 6-177 to 6-178
OA$FBT_REMOVE_LIBRARY, 6-179 to 6-180
OA$FBT_WRITE_LIBRARY, 6-181 to 6-182
OA$FLD_NEXT_LIST, 6-188
OA$FLD_nnn, 6-183 to 6-187
See also individual functions
OA$FLD_RECOGNITION, 6-189
OA$FLD_SCROLL_OFF, 6-190
OA$FLD_SEARCH, 6-191
OA$FLO_CLEAR_VA, 6-192
OA$FLO_CLOSE_LIB, 6-193 to 6-194
OA$FLO_OPEN_LIB, 6-195 to 6-196
OA$FRM_SET_FIELD, 6-197
OA$HAR_CHECK_DECCRT, 6-198 to 6-199
OA$INI_GLOBALS, 6-201
OA$INI_INITIALIZE, 6-202 to 6-203
OA$INI_LOGICALS, 6-204
OA$INI_OPEN_LIB, 6-205
OA$INI_SET_DEFAULT_DIR, 6-206
OA$INI_SUBPROCESS, 6-207 to 6-208

Functions (Cont.)
OA$KEY_COMMAND, 6-209
OA$LIST_xxx, 6-210 to 6-211
OA$LOG_DISPLAY, 6-212 to 6-213
OA$LOG_TO_FILE, 6-214 to 6-215
OA$MENU_LEVEL_PUSH, 6-216 to 6-217
OA$MENU_SHOW_STACK, 6-218 to 6-219
OA$MRG_TTOUT, 6-220 to 6-221
OA$MSG_PURGE, 6-222
OA$MSG_TRAP, 6-223
OA$SCL_nnn, 6-224 to 6-227
See also individual functions
OA$SCP_STACK_DUMP, 6-228
OA$STAT_GET_STATISTICS, 6-229 to 6-230
OA$SUB_CLOSE, 6-231
OA$SUB_OPEN, 6-232
OA$SYM_CLOSE, 6-233
OA$SYM_GET_SYMBOL, 6-234 to 6-235
OA$SYM_MAKE_SYMBOL, 6-236 to 6-237
OA$SYM_OPEN, 6-238 to 6-239
OA$TRA_OUT, 6-240
OA$TRA_SET, 6-241 to 6-243
OA$TRM_BRDCST_INIT, 6-244 to 6-244.1
OA$TRM_BRDCST_STOP, 6-244.2 to 6-244.3
OA$TRM_INIT, 6-245
OA$TRM_INQUIRE, 6-246
OA$TRM_PORT_LINE, 6-247
OA$TRM_PORT_LOG_ON, 6-248
OA$TRM_PORT_OFF, 6-249
OA$TRM_PORT_ON, 6-250
OA$TRM_PORT_PRINT, 6-251

Functions (Cont.)
OA$TXL_CLOSE, 6-252
OA$TXL_COMPILE, 6-253
OA$TXL_DIRECTORY, 6-254
OA$TXL_INSTALL, 6-255
OA$TXL_LIST, 6-256
OA$TXL_OPEN, 6-257 to 6-258
OA$TXL_REMOVE, 6-259
OA$VAL_SET_VALID, 6-260 to 6-261
PARSE_USER, 6-262 to 6-263
pipelining, 6-9
source function, 6-9
PORT_OFF, 6-264
PORT_ON, 6-264
PRINT_SCREEN, 6-265 to 6-266
PROMPT, 6-267 to 6-270
PROMPT_ID, 6-271 to 6-272
PURGE_FILE, 6-273 to 6-276
PUT_FIELD, 6-277 to 6-278
QUEUE_BATCH, 6-279 to 6-283
QUEUE_PRINT, 6-284 to 6-288
RENAME, 6-289 to 6-290
REPEAT, 6-291 to 6-292
SAVE, 6-293 to 6-295
SCRIPT, 6-296 to 6-297
SET_DEFAULT, 6-298 to 6-299
SET_MENU, 6-300 to 6-301
SHOW_FORM, 6-302
SORT_LIST, 6-303 to 6-305
SPAWN, 6-306 to 6-307
stacking, 6-4 to 6-9
See also Function lists
STATISTICS, 6-308 to 6-309
TEST_SPEC, 6-310 to 6-311
TIME, 6-312 to 6-313
trapping errors from, 6-6
types of, 6-1
UNWIND, 6-314 to 6-315
user-defined, 6-10 to 6-11

Functions (Cont.)
restriction on implementing, 6-10
calling standards, 16-22
VAX/VMS requirements, 16-21
VERSION, 6-316 to 6-317
WAIT, 6-318 to 6-320
WPSPLUS$CBI_CLOSE_WINDOW, 6-321
WPSPLUS$CBI_CLOSE_WINDOW2,6-322
WPSPLUS$CBI_OPEN_WINDOW, 6-323 to 6-324
WPSPLUS$CBI_OPEN_WINDOW2, 6-325 to 6-326
WPSPLUS_SPELL, 6-327
WRITE, 6-328 to 6-329
YESNO_PROMPT, 6-330 to 6-332
YESNO_PROMPT_ID, 6-333 to 6-334

G

Generic menu design, 5-1 to 5-8
/GET form qualifier, 3-14
and /GET_SAVE field qualifier, 3-15
processed after /PRE_FUNCTION, 3-14
GET function, 6-101 to 6-107, 16-17
and complex DSRs, 6-107
from the subprocess, 7-11, 7-12
GET OA$DCL, 7-19
GET OA$FUNCTION
special GET destination and SAVE, 6-294
/GET_SAVE field qualifier, 4-16 to 4-18
See also /GET
and /GET form qualifier, 3-16
Gold BACKSPACE
previous page in a form set, 3-80

Gold TAB
next page in a form set, 3-80
Gold-key sequences
and Recognition, 4-46
in a scrolled region, 4-34
in menu choice field, 3-31
predefined, 4-56 to 4-63

H
Hard Copy Mode, 3-16, B-1 to B-5
See also MODE function
providing information while in, 4-18
/HARD field qualifier, 4-18 to 4-19
/HARD form qualifier, 3-16
Help
and Gold H, 4-59
facility, 14-1 to 14-22
See also HELP function
additional topics, 14-14 to 14-18
HELP forms, 14-5
keyboard-style help, 14-5
library, 14-6
cross-references, 14-14 to 14-18
line-1 (direct), 14-17
modifying, 14-18
module format (menu), 14-10
module format (nonmenu), 14-6
restrictions, 14-20
switching to another, 14-21
Help form
.TYPE HELP, 3-6
/HELP form qualifier, 3-17, 14-2
HELP function, 6-108 to 6-111
backing off, 14-4
called after DISPLAY, 6-61
called from a prompt, 6-269
context checking, 14-2
and choice-field contents, 14-3
from a prompt, 14-3, 14-22
from the editor, 14-3
Help function
context checking (Cont.)
on a menu, 14-2
on a nonmenu, 14-2
with /HELP, 14-2
from the default editor, 14-5
syntax restrictions, 6-110
/HIGHLIGHT form qualifier, 3-17
performance impact, 3-18
used with FMS field highlighting, 3-18

I
IFEXIT function, 6-112 to 6-113
in function lists, 6-6
INCLUDE function, 6-114 to 6-115, 10-16
Indexed data files
See Data files
Indexed files, RMS
defined by entry forms, 3-37
Initialization and flow control, 2-1 to 2-6
partial, 2-3
standard sequence of, 2-2
Input Processing facility, 3-31
Installation Verification Procedure
See IVP
Interrupt menu
and Gold I, 4-60
/INVALID field qualifier, 4-4, 4-19 to 4-22
and /RECOGNITION, 4-21
on entry form key fields, 3-46
Iterative symbol substitution
in FOR-function calls, 6-90

K
/KEY entry form qualifier, 3-50
and /BEGIN, 3-51
and /DATA, 3-48, 3-50
and /FIELDS, 3-49, 3-51
and /SAVE_START, 3-52
highlighting by, 3-51

Key of reference, RMS
in an FDL-created file, 3-39
Key, NEXT LIST
and /LIST field qualifier, 4-60
keypad ., 4-60
Keyboard directives
See also individual directives
recognized at prompts, 6-269
where they are recognized, 4-63
Keyboard keys
default definitions of, 4-56 to 4-63
redefined by /SCROLL, 4-34
Keyboard, defining/redefining the, 4-63 to 4-65
/KEYPAD form qualifier, 3-18
KEYPAD function, 6-116 to 6-117
Keypad keys
See also Terminator key(s)
defining/redefining, 4-63 to 4-65
0 — EXIT SCREEN key, 4-59, 6-113
— NEXT LIST key, 4-60
reserved keys, 4-65
/KEYS entry form qualifier
ignored during file creation, 3-38
Keys, data file
ALL-IN-1 versus RMS, 3-39
specified with FDL, 3-39

L

Layered products
ACMS, 6-13 to 6-15
DATATRIEVE, 6-65
Left Brace character
used as script mnemonic delimiter, 15-3
Libraries, form, 3-2, 16-9
opening/reopening
with NEWLIB, 6-152 to 6-154
LIGHTS function, 6-118 to 6-119
/LIST field qualifier, 4-22 to 4-23
and /LIST select form qualifier, 3-70
List form
.TYPE LIST, 3-6
LIST function, 6-120 to 6-123
as a pipeline source, 6-10
LIST key
Gold L, 4-57
List processing, 10-21 to 10-27
See also MERGE, MERGE_LIST
with MERGE, 6-134 to 6-140
with MERGE_LINE, 6-141 to 6-142
WPS-PLUS
MERGE_LIST, 6-143 to 6-146
SORT_LIST, 6-303 to 6-305
TEST_SPEC, 6-310 to 6-311
/LIST select form qualifier, 3-69
and /LIST field qualifier, 3-70
and SEL_STYLE subfunction, 3-70
Local symbol
table, 27
Local symbols, 8-4
/LOG form qualifier, 3-19
LOG function, 6-124 to 6-125
LOGICAL function, 6-126 to 6-128
from the subprocess, 7-12
Logical name, 27
See also Logicals
Logicals, 7-16 to 7-18
main process
and LOGICAL function, 6-126 to 6-128
main process versus subprocess, 7-17

M

/MAIL form qualifier, 3-19
MAIL function, 6-129
See also Electronic Messaging
subfunctions, 11-10 to 11-51
ANSWER, 11-11
ATTACH, 11-13
AUTO_FORWARD, 11-13
AUTO_REPLY, 11-14

MAIL function (Cont.)
CANCEL, 11-15
CANCEL_AUTO_REPLY, 11-15
CC, 11-15
CLOSE_MESSAGE, 11-16
CREATE, 11-16.1
DEFAULT_MESSAGE, 11-19
DEFER, 11-20
DELETE, 11-20
DELIVERY_RECEIPT, 11-21
DETACH, 11-22
DIST_COPY, 11-22
DIST_CREATE, 11-23
DIST_DELETE, 11-23
DIST_EDIT, 11-24
DIST_LIST, 11-24
EDIT, 11-25
EXCHANGE_CURMES, 11-26
EXPAND_DIST_LIST, 11-26
FILE_ATTACHMENT, 11-27
FILE_BODY, 11-28
FILE_MESSAGE, 11-29
FORWARD, 11-30
FORWARDABLE, 11-32
FUNCTION, 11-32
GET, 11-33
INDEX, 11-33
MAIL_DOCUMENT, 11-34
NEXT, 11-36
NO_DELIVERY_RECEIPT, 11-37
NON_FORWARDABLE, 11-37
ORIGINAL, 11-37
OUTBOX, 11-38
POP_CURMES, 11-38
PRINT, 11-39
PRIORITY, 11-40
PUSH_CURMES, 11-41
MAIL function (Cont.)
READ, 11-42
READ_RECEIPT, 11-43
SEND, 11-44
SHOW, 11-46
STATUS, 11-46
SUBJECT, 11-47
TEXT, 11-48
TO, 11-48
VAXMAIL_CHECK, 11-49
Mailboxes, 7-3
and the subprocess, 7-2
use of in applications, 16-20
MAKE_FILE_NAME function, 6-130 to 6-133
from the subprocess, 7-11
Mathematical calculations
operators and operands for, 6-38
with COMPUTE, 6-37 to 6-41
with the Desk Calculator form, 3-58
Mathematical computations
setting decimal places for, 4-11, 6-54 to 6-55
with /COMPUTE field qualifier, 4-7
with Gold #, 4-58
Menu form, 3-24 to 3-36
choice field processing, 3-29
creating a complex, 3-33 to 3-36
Named Data, 3-25
processing
flow control, 3-24
qualifiers
/CAPTIVE, 3-26
/CHOICE (required), 3-25
/FIELDS, 3-26
/MORE, 3-27
/ONCE, 3-28
required features, 3-24
selecting the current item, 4-58, 4-62
selecting the first item, 4-62
.TYPE MENU, 3-6

Menu form processing, 3-28
  complex menus, 3-33 to 3-36
Menu levels, 3-36
Menu options, stacking, 3-32
  restrictions when, 3-33
Menu stack, prior, 3-35
Menu stacks, 3-36
MERGE function, 6-134 to 6-140
  as a pipeline source, 6-10
  examples, 10-22 to 10-27
  flow control, 10-25, 10-27
  header file
    and OA$PAGE_LENGTH, 6-136
  header-file, 10-26
  list file, 10-24
    and OA$MERGE_KEY, 6-136
  template directives, 6-137 to 6-139
    continuation lines, 6-139
MERGE_LINE function, 6-141 to 6-142
  template directives with, 6-141
MERGE_LIST function, 6-143 to 6-146
  See also SORT_LIST, TEST_SPEC
Message buffer
  viewing
    with Gold W, 4-60
Message facility, 14-22 to 14-26
  See also OAMESS.MSG
  and Help, 14-26
  message buffer, 14-24
    viewing, 14-25
  message severity, 14-24
  message syntax, 14-23
MESSAGES key
  Gold W, 4-60
Mnemonic delimiters
  script, 15-3
Mnemonic, scripts, 15-3
/MODE entry form qualifier, 3-44
MODE function, 6-147 to 6-149
Modes
  Command (Hard Copy)
    providing information while in, 4-18
.MORE, 3-7
.MORE directive, 3-7
/MORE menu qualifier, 3-26, 3-27
  in FORM function calls, 3-9
  sample application of, 3-34
Multinode features
  overview, 12-1
Multipage form
  See Form set

N

Named Data, 3-28, 30
  dynamically redefining, 3-75
    with /RESET, 3-77
  modifying, 16-6
  reorganizing, 16-9
  searching for, 3-7
  use of by ALL-IN-1, 3-4
  viewing
    with Gold N, 4-60
Nesting functions
  See FOR, MERGE, SCRIPT
NEWDIR function, 6-150 to 6-151
NEWLIB function, 6-152 to 6-154, 16-11
/NEXT form qualifier, 3-19
  and the prior menu stack, 3-75
NEXT LIST key
  and /LIST field qualifier, 4-60
  keypad ., 4-60
NEXT PAGE key
  Gold TAB, NEXT SCREEN, 4-60
NEXT_LIST function, 6-155 to 6-156
  from the subprocess, 7-11
$NEXT_LIST_ITEM
  maintained with NEXT_LIST, OA$FLD_NEXT_LIST, 7-11

Nonstandard form, 3-80
/NOWRITE entry form qualifier, 3-52

O
&OA
function-call prefix, 6-2, 6-137
OA
function-call prefix, 6-2, 7-5
OA$BUILD:OASDF.BLI
and user-defind functions 6-11
OA$BUILD:HLPSOURCE.HLB
Help-text sources, 14-18
OA$CLI_GET_SYMBOL
function, 6-157 to 6-158
OA$CLI_GET_VALUE
function, 6-159 to 6-160
OA$CNV_CONVERT function, 6-161 to 6-162
OA$CNV_DIST_LIST
function, 6 163 to 6-164
OA$CNV_FROM_ _SCROLL
function, 6-167, 6-224 to 6-227
See also /SCROLL
OA$CNV_MERGE function, 6-165 to 6-166
OA$CNV_TO_SCROLL
function, 6-168, 6-224 to 6-227
See also /SCROLL
OA$CURDOC
special symbol
and CABINET CLEAR_CURDOC, 9-23
and CABINET COPY, 9-25
and CABINET CREATE, 9-26
and CABINET CROSSFILE, 9-26
and CABINET CURRENT, 9-27
and CABINET DELETE_OR_REFILE, 9-29
and CABINET NEXT_FOLDER, 9-36
and CABINET NEXT_NUMBER, 9-37
and CABINET NEXT_TITLE, 9-40
and CABINET SELECT, 9-45
OA$DATA
and LOGICAL function, 6-128
subprocess versus main process, 7-17
OA$DATA:ATTENDEE.DAT
system attendee file, 13-1
OA$DATA:CALACCESS.DAT
CALENDAR access file, 13-1
OA$DATA:DAF.DAT
See SDAF
OA$DATA:FORMAT.DAT
Format master file, 10-28
OA$DATA:MEETING.DAT
system meeting file, 13-1
OA$DATA:PENDING.DAT, 9-12
and CABINET GET_PENDING, 9-31
and CABINET PUT_PENDING, 9-42
system Pending file, 9-2
OA$DATA:username.A1CAL
the personal calendar file, 13-1
OA$DATE
special symbol
and DATE_CONVERT, 6-48
validation DSR, C-2
OA$DATE_FULL
special symbol
and DATE_CONVERT, 6-48
OA$DATE_NBS
special symbol
and DATE_CONVERT, 6-48
OA$DCL
special GET destination, 6-52, 16-23
OA$DEFAULT_EXTENSION
special symbol, 6-131, 10-3
used to create file name, 4-12
OA$DIR
field-name keywords, C-3
special DSR, C-3

OA$DISPLAY
special GET destination, 6-61
OA$DSA__CACHE__STATUS
function, 6-70, 6-169 to 6-170
OA$EDIT__BUFFER
text data set, 10-10
OA$EDIT__FILE
special symbol, 6-71
OA$EDIT__INI__FILE
special symbol, 6-72
OA$EDIT__PURGE__COUNT
special symbol, 6-72
OA$EDITOR
special symbol, 6-73
OA$ENT__SET__xxx function,
6-171 to 6-172
OA$FBT__INSTALL__LIBRARY
function, 6-173 to 6-174
OA$FBT__LIST__LIBTREE
function, 6-175 to 6-176, 6-194,
6-196
OA$FBT__LIST__TREE
function, 6-177 to 6-178
OA$FBT__REMOVE__LIBRARY
function, 6-179 to 6-180
OA$FBT__WRITE, 16-11
OA$FBT__WRITE__LIBRARY
function, 6-181 to 6-182
OA$FIELD__TERM
special symbol, 7-13
and BACKSPACE, 4-57
and Gold BACKSPACE,
PREV SCREEN, 4-61
and Gold TAB, NEXT
SCREEN, 4-61
and keypad 0, 4-59
and PROMPT, 6-267, 6-271
and RETURN, 4-58
and TAB, 4-62
OA$FLD__BOTTOM function,
4-67, 6-183
and Gold B, 4-57
OA$FLD__DONE function,
4-67, 6-184
and PROMPT, 6-267, 6-271
and RETURN, Gold F, 4-58
OA$FLD__DOWN function, 4-67,
6-184
and Down Arrow, 4-58
OA$FLD__EXIT function,
4-67, 6-184
and IFEXIT, 6-113
and keypad 0, Gold K, Gold
Q, 4-59
and PROMPT, 6-267, 6-271
and the prior form, 3-75
OA$FLD__NEXT function, 6-184
and TAB, 4-62
OA$FLD__NEXT__LIST
function, 4-67, 6-188
and /LIST field qualifier, 4-60
and the NEXT LIST key, 4-60
from the subprocess, 7-11
OA$FLD__nnn functions, 4-66,
6-183 to 6-187
See also individual functions
OA$FLD__PAGE__BACK
function, 4-67, 6-185
and Gold BACKSPACE, PREV
SCREEN, 4-61
OA$FLD__PAGE__BOTTOM
function, 6-185
OA$FLD__PAGE__FORWARD
function, 4-67, 6-185
and Gold TAB, NEXT
SCREEN, 4-60
OA$FLD__PAGE__TOP
function, 6-185
OA$FLD__PREV function, 4-67,
6-185
and BACKSPACE, 4-56
OA$FLD__RECOG function
and Gold L, 4-57
OA$FLD__RECOGNITION
function, 6-189
OA$FLD__SCROLL__OFF
function, 6-190
OA$FLD__SEARCH function,
6-191
and Gold S, 4-57

OA$FLD_TOP function, 4-67
and Gold T, 4-62
OA$FLD_UP function, 4-67, 6-186
and Up Arrow, 4-62
OA$FLO_CLEAR_VA function, 6-192
OA$FLO_CLOSE function
and DUMP_CACHE, 6-70
OA$FLO_CLOSE_LIB function, 6-193 to 6-194
and DUMP_CACHE, 6-194
and NEWLIB, 6-194
OA$FLO_OPEN function, 16-11
and DUMP_CACHE, 6-70
OA$FLO_OPEN_LIB function, 6-195 to 6-196
and DUMP_CACHE, 6-196
and NEWLIB, 6-196
OA$FORM_DISPOSE
special symbol
See also Form dispositions
and CABINET, 6-6
and OA$FLD_BOTTOM, 4-57
and OA$FLD_DONE, 4-58
and OA$FLD_EXIT, 4-59
and OA$FLD_NEXT, 4-62
and OA$FLD_PAGE_BACK, 4-61
and OA$FLD_PAGE_FORWARD, 4-61
and OA$FLD_PREV, 4-57
and OA$FLD_TOP, 4-62
and YESNO_PROMPT, 6-330, 6-334
set by OA$FLD_nnn functions, 4-67
summary of values for, 4-68
OA$FORM_TERMINATOR
special symbol, 7-13
and BACKSPACE, 4-57
and Gold BACKSPACE, PREV SCREEN, 4-61
and Gold TAB, NEXT SCREEN, 4-61
and keypad 0, 4-59
and PROMPT, 6-267, 6-271
and RETURN, 4-58
and TAB, 4-62
and YESNO_PROMPT, 6-330, 6-334
terminator values, 4-69
OA$FRM_SET_FIELD function, 6-197
OA$FUNCTION
special GET destination
and ERROR, 6-77
and LIST, 6-122
and LOG, 6-125
and LOGICAL, 6-128
and MERGE, 6-140
and NEWDIR, 6-151
and PURGE_FILE, 6-275
OA$FUNCTION, GET
special GET destination
and SAVE, 6-294
OA$HAR_CHECK_DECCRT function, 6-198 to 6-199
OA$HELP_BOTTOM
HELP form, 14-5
OA$HELP_TOP
HELP form, 14-5
OA$INI_GLOBALS function, 6-201
OA$INI_INITIALIZE function, 6-202 to 6-203
OA$INI_LOGICALS function, 6-204
OA$INI_OPEN_LIB function, 6-205
OA$INI_SET_DEFAULT_DIR function, 6-206

OA$INI_SUBPROCESS
function, 6-207 to 6-208
OA$KEY_COMMAND
function, 6-209
OA$LIB
and LOGICAL function, 6-128
subprocess versus main process, 7-17
OA$LIB:ACTITEM.FDL, 3-42
used to create OAUSER:ACTITEM.DAT, 3-42
OA$LIB:FINDCOM.COM
and SET_DEFAULT, 6-299
OA$LIB:MEMRES.FLB
memory-resident form library, 3-2, 16-10
OA$LIB:OAFORM.FLB
altering search order of, 3-3
common form library, 3-2
OA$LIB:OAHELP.HLB
See also Help library
ALL-IN-1 help library, 6-110
default help library, 14-7
OA$LIB:STANDARD.FLB
prototype form library, 3-2
OA$LIB:USER.FLB
user's form library, 3-2
OA$LIST_xxx function, 6-210 to 6-211
OA$LOG_DISPLAY function, 6-212 to 6-213
OA$LOG_TO_FILE function, 6-214 to 6-215
OA$MAIDES
validation DSR, C-7
OA$MAIL_ADD_ADDR
special validation DSR, 11-50
OA$MAIL_ADDRESS
special validation DSR, 11-50
validation DSR, C-8
OA$MAIL_COUNT
special symbol
and CABINET GET_PENDING, 9-32
OA$MAIL_NUM_ADDRESSES
MAIL-related logical, 11-9
OA$MAILPRIO
special validation DSR, 11-50
validation DSR, C-8
OA$MENU_LEVEL_PUSH
function, 6-216 to 6-217
OA$MENU_REMAINDER
special GET destination, 3-33
OA$MENU_SHOW_STACK
function, 6-218 to 6-219
OA$MERGE_KEY
special symbol
and MERGE, 6-136
and MERGE list file, 10-25
OA$MERGE_LINE
special GET destination, 10-26
and MERGE_LINE, 6-142
OA$MERGE_REC
See OA$MERGE_KEY
OA$MRG_TTOUT function, 6-220 to 6-221
See also MERGE, MERGE_LINE
and MERGE, 6-135
and MERGE_LINE, 6-142
OA$MSG_FAC
special symbol, 14-25
OA$MSG_ID
special symbol, 14-25
OA$MSG_PURGE function, 6-222
OA$MSG_SEV
special symbol, 14-25
OA$MSG_TEXT
special symbol, 14-25
OA$MSG_TRAP function, 6-223
OA$PAGE_LENGTH
special symbol
and MERGE, 6-136, 10-27
OA$PAGE_NUMBER
special symbol
and MERGE, 6-136, 10-27
OA$PROMPT_HELP
special symbol, 14-3
and %OA-I-LASTLINE, 6-62
and HELP, 6-269
and HELP function, 6-111
and prompts, 6-269

OA$PROMPT_TEXT
special symbol
and PROMPT, 6-267, 6-271
OA$SCL_BOTTOM function
and Gold Down Arrow, 4-34
OA$SCL_DOWN function
and the Down Arrow, 4-34
OA$SCL_FIRST_PAGE function
and Gold T, 4-34
OA$SCL_LAST_PAGE function
and Gold B, 4-34
OA$SCL_NEXT_PAGE function
and Gold TAB, 4-34
OA$SCL_nnn functions, 6-224
to 6-227
See also individual functions
OA$SCL_PRIOR_PAGE function
and Gold BACKSPACE, 4-34
OA$SCL_TOP function
and Gold Up Arrow, 4-34
OA$SCL_UP function
and the Up Arrow, 4-34
OA$SCP_STACK_DUMP
function, 6-228
OA$SCRIPT_EXIT_HANDLER
special symbol
and Gold X, 15-4
and Gold X, CTRL/C, 4-59
OA$SCRIPT_PSIB
script pseudo-input buffer, 15-3
and script text lines, 15-3
in script DO Mode, 15-4
OA$SCRIPT_TERM
script pseudo-input buffer
terminator, 15-4
OA$SCRIPT_TEXT
script pseudo-input buffer
text, 15-4
OA$SEL_ADDR
special symbol
used with FOR function, 3-73
OA$SEL_COUNT
special symbol
used with FOR function, 3-73
OA$SEL_DSAB
special symbol
used with FOR function, 3-73
OA$SEL_KEY
special symbol
and validation post-
functions, 4-51
used with FOR function, 3-74
OA$SEL_LINE
special symbol
used with FOR function, 3-74
OA$SHARE_CUR
and CABINET
SHARE_DOCUMENT, 9-47
OA$SHARExxx logicals, 9-19
See also File Cabinet shared
area
OA$SRC_COUNT
special symbol
used with FOR function, 3-74
OA$STAT_GET_STATISTICS
function, 6-229 to 6-230
OA$STATUS
special symbol
and CABINET, 6-6
and CABINET
NEXT_FOLDER, 9-36
and CABINET subfunction
calls, 9-15
OA$SUB_CLOSE function, 6-231
OA$SUB_OPEN function, 6-232
OA$SYM_CLOSE function, 6-233
OA$SYM_GET_SYMBOL
function, 6-234 to 6-235
OA$SYM_MAKE_SYMBOL
function, 6-236 to 6-237
OA$SYM_OPEN function, 6-238
to 6-239
OA$TABLE
validation DSR, C-8
OA$TRA_OUT function, 6-240
OA$TRA_SET function, 6-241
to 6-243
OA$TRM_BRDCST_INIT
function, 6-244 to 6-244.1

OA$TRM_BRDCST_STOP
function, 6-244.2 to 6-244.3
OA$TRM_INIT function, 6-245
OA$TRM_INQUIRE function, 6-246
OA$TRM_PORT_LINE
function, 6-247
OA$TRM_PORT_LOG_ON
function, 6-248
OA$TRM_PORT_OFF
function, 6-249
OA$TRM_PORT_ON function, 6-250
OA$TRM_PORT_PRINT
function, 6-251
OA$TXL_CLOSE function, 6-252
OA$TXL_COMPILE function, 6-253
OA$TXL_DIRECTORY
function, 6-254
OA$TXL_INSTALL function, 6-255
OA$TXL_LIST function, 6-256
OA$TXL_OPEN function, 6-257 to 6-258
OA$TXL_REMOVE function, 6-259
OA$TXL_SEARCH_LAST, 15-6
OA$VAL_SET_VALID
function, 4-50, 4-54, 6-260 to 6-261
OA$VIEW_FIELDS
View Fields form, 4-63, 6-149, 16-18
OA$VIEW_MESSAGES
and Gold H, 4-60
and message buffer, 14-25
View Messages form, 4-60, 16-18
OA$VIEW_ND
View Named Data form, 4-60, 6-149, 16-18
OA$YESNO
validation DSR, C-9
OA$YN
validation DSR, C-9
OAHELP
logical name, 14-21
main-process logical, 6-110
OAINI.COM
BYE function redefined in, 6-24
EXIT function in, 6-79
subprocess log-in file, 7-2
OAMAILBOX
and DCLMAILBOX, 7-4
calling multiple functions through, 6-8
mailbox from subprocess, 7-4
multiple writes to, 7-7
syntax while writing to, 7-6
OAMESS.MSG
message file, 14-23
updating the, 14-23
OAUSER
and COMMAND function, 6-36
and DCL function, 6-53
and SET_DEFAULT, 6-299
subprocess versus main process, 7-17
OAUSER:ACTITEM.DAT
file structure, 3-40
OAUSER:CALACCESS.DAT
Action Items file, 13-1
OAUSER:DAF.DAT
See PDAF
OAUSER:DOCDB.DAT
personal File Cabinet index, 9-2, 9-4
keys to, 9-4, 9-5
reorganizing
and CABINET REORG_DOCDB, 9-44
OAUSER:PASTE.TMP
See Scratch Pad
/ONCE form qualifier
in entry forms, 3-52
/ONCE menu qualifier, 3-28
/ONE_ENTRY entry form qualifier, 3-53
and /SAVE_START, 3-53
/OPTIONAL field qualifier, 4-21, 4-23 to 4-24, 4-49

/OVERLAY form qualifier, 3-20

P

Pad, scratch
  and Gold *, 4-61
PARSE_USER function, 6-262 to 6-263
  from the subprocess, 7-11
PDAF, 9-2, 9-9, 9-46
  and OAUSER:DOCDB.DAT, 9-9
Performance
  and /HIGHLIGHT, 3-18
  and FMS field highlights, 16-23
  and RMS tuning, 16-23
  and symbol types, 8-15, 16-23
  command procedures versus scripts, 16-22
  effect of stacking functions on, 6-8
Permanent symbol, 34
Permanent symbol table, 34
  username.PST, 8-31
Permanent symbols, 8-2, 8-31 to 8-36
Personal Document Attributes File
  See PDAF
Pipeline mechanism, 6-9
  source functions
    MERGE, 6-134, 6-141
  with COPY, 6-44
PORT_OFF function, 6-264
PORT_ON function, 6-264
  and PRINT_SCREEN, 4-61
/POST_FUNCTION
  appended in FORM function calls, 3-9
/POST_FUNCTION field qualifier, 4-24 to 4-25
  and the prior menu stack, 3-75
  versus /POST_FUNCTION form qualifier, 4-24
/POST_FUNCTION form qualifier, 3-20, 3-21
  and the prior menu stack, 3-75
/PRE_FUNCTION
  appended in FORM function calls, 3-9
/PRE_FUNCTION field qualifier, 4-25 to 4-26
  and the prior menu stack, 3-75
/PRE_FUNCTION form qualifier, 3-21
  and the prior menu stack, 3-75
Precompiling forms, 16-11
  with the PCF option, 16-11
PREVIOUS PAGE key
  Gold BACKSPACE, PREV SCREEN, 4-61
PRINT SCREEN key
  and PRINT_SCREEN function, 4-61
  Gold P, 4-61
PRINT_SCREEN
  saving in PASTE.TMP, 4-61
PRINT_SCREEN function, 6-265 to 6-266
  from the editor, 4-61
  Gold P, 4-61
Printing files
  See QUEUE_PRINT
Prior menu stack, 3-35
Profile
  See User Profile
Profile, user
  privileges
    PRVCMD, 3-23, 3-32
    PRVDCL, 3-32
    PRVSUP, 3-23
Programming Functions, 16-12 to 16-14
/PROMPT field qualifier, 4-26 to 4-27
PROMPT function, 6-267 to 6-270
  and OA$FLD_nnn functions, 4-67
  and user-defined keys, 6-269
  from the subprocess, 7-13

PROMPT_ID function, 6-271 to 6-272
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13
PRVCMD
User Profile privilege, 3-23, 3-32, 6-2
PRVDCL
User Profile privilege, 3-32, 4-58
PRVSUP
User Profile privilege, 3-23
PURGE_FILE function, 6-273 to 6-276
PUT_FIELD function, 6-277 to 6-278
/PUT_SAVE field qualifier, 4-27 to 4-28

Q

Qualifiers
See also Field qualifiers, Form qualifiers
Field versus Form, 3-10
Qualifiers, field
used to define a field, 4-2
QUEUE_BATCH function, 6-279 to 6-283
versus COMMAND, 6-282
QUEUE_PRINT function, 6-284 to 6-288
Quit form
See EXIT SCREEN

R

Recognition
full
and Gold S, 4-57
partial
and Gold L, 4-57
with /INVALID, 4-19
with /RECOGNITION, 4-28
with /VALID, 4-47
/RECOGNITION field qualifier, 4-28 to 4-30
with /VALID, 11-50
Record collection
controlling output of, 4-55 to 4-56
with /USE_FORM, 4-55
creating
with FOR function, 6-83 to 6-91
with select form, 3-68, 3-71
with validation qualifiers, 4-55
displaying
with SEL_DISPLAY subfunction, 3-72, 6-87
formatting
with /SHOW, 4-45
with /STYLE select-form qualifier, 3-70
with search form .SOUT directive, 3-67
with SEL_STYLE subfunction, 3-72, 6-87
number of records in
OA$SEL_COUNT, 3-73
putting unique records into
with /UNIQUE, 4-47
refining
with FOR FIRST, 6-84
with FOR UNIQUE, 6-84
saving
with /LIST select-form qualifier, 3-69
selecting from
with /STYLE select-form qualifier, 3-70
with OA$SEL_ADDR, 3-73
with OA$SEL_KEY, 3-74
with OA$SEL_LINE, 3-74
with SEL_CHOICE subfunction, 3-72, 6-87
source of records in a
OA$SEL_DSAB, 3-73
viewing
with Gold L, 4-57
with Gold S, 4-57

Record Management Services
See RMS
Record selection expression
See RSE
Redefining the keyboard, 4-63 to 4-65
Relational operators, 6-85
RENAME function, 6-289 to 6-290
REPEAT function, 6-291 to 6-292
in function lists, 6-7
/RESET form qualifier, 3-22
in FORM function calls, 3-9
using the, 3-77
RESULT
common DCL symbol, 7-12
and LOGICAL, 6-126
updating, 6-102
DCL symbol
set by OAFLD_DONE, OAFLD_EXIT, 4-67
Retrieving symbol values
with /GET_SAVE, 4-16
RETURN key
and OA$FLD_DONE, 4-58
Right Brace character
used as script mnemonic delimiter, 15-3
RMS indexed files
defined by entry forms, 3-37
RMS key of reference
See also Data file keys
in an FDL-created file, 3-39
RMS tuning
for improved performance, 16-23
RSE
differences between search and select forms, 3-61
in field validation, 4-30 to 4-32
in search form .TYPE entry, 3-63
in the FOR function, 6-83
/RSE_INVALID field qualifier, 4-4, 4-30 to 4-32
/RSE_RECOG field qualifier, 4-30 to 4-32
/RSE_VALID field qualifier, 4-4, 4-30 to 4-32

S

SAVE function, 6-293 to 6-295
from the subprocess, 7-11
/SAVE_START entry form qualifier, 3-53
and /KEY key fields, 3-53
and /ONE_ENTRY, 3-53
Scratch Pad
and Gold*, 4-61
SCRIPT function, 6-296 to 6-297
See also DO
Script processing
uses of scripts, 15-1
Scripts, 15-1 to 15-34
See also SCRIPT, DO functions
adding comments to, 15-25
block output
beginning, 15-10
calling functions from, 15-14, 15-15
conditional processing in, 15-17
controlling cursor location in, 15-12, 15-13, 15-26
controlling screen appearance in, 15-11, 15-12, 15-24, 15-26, 15-30
debugging, 15-29
directives
.ABORT, 15-4, 15-9
as functions, 15-31
.BLOCK, 15-10
.CHECK_FIELD, 15-10
.CHECK_FORM, 15-11
.CLEAR, 15-11
.CUR_KEEP, 15-12
.CUR_RESTORE, 15-13
.CUR_SAVE, 15-13
.CURSOR, 15-12
.DISPLAY, 15-13
.EXIT, 15-4, 15-14
.FUNCTION, 15-14

Scripts (Cont.)
.FX, 15-15
.FZ, 15-15
.GET_FIELD, 15-16
.GOTO, 15-17
.IF, 15-17
.JUDGE, 15-19
.LABEL, 15-21
.PAUSE, 15-21
.PROCESS, 15-22
and OA$SCRIPT_PSIB, 15-3
.PROMPT, 15-24, 15-28
.REFRESH, 15-24
.REMARKS, 15-25
.RESET_CURSOR, 15-26
.SET_PAUSE, 15-26
.SET_REFRESH, 15-26
.SHOW_FORM, 15-27
.SINX, 15-28
.TEXT, 15-29
.TRACE, 15-29
.TRACE_ON, 16-15
.VIDEO_ATT, 15-30
.WAIT, 15-30
displaying forms in, 15-27
example, 15-32
exiting from, 4-59, 15-4, 15-9, 15-14
field processing in, 15-16
flow control in, 15-17, 15-19, 15-21
format, 15-7
graphics in, 15-13
interactive directives
and OA$SCRIPT_PSIB, 15-3
mnemonic delimiters, 15-3
mnemonics for input, D-1
nested, 15-2
output from, 15-29
pausing in, 15-21, 15-26
processing input while in, 15-22
prompting from, 15-24, 15-28, 15-30
synchronizing, 15-10, 15-11
text lines, 15-8
terminators in, 15-8
tracing
with Gold (, 6-34
two modes of, 15-2
DO, 15-2
mixing the, 15-2
Scripts mnemonic, 15-3
Scripts, command procedures
versus performance in applications, 16-22
/SCROLL field qualifier, 4-33 to 4-44, 6-224
Scrolled region
creating with FMS, 4-34
identifying with /SCROLL, 4-33
navigating in
with OA$SCL_nnn functions, 6-224
Scrolling
display-only, 4-43
Scrolling functions
See OA$SCL_nnn
SDAF, 9-2, 9-9, 9-46
and OAUSER:DOCDB.DAT, 9-9
Search form, 3-60 to 3-68
and FOR function, 3-65
Named Data, 3-61
.SFILE directive, 3-65
.SOUT directive, 3-67
processing, 3-68
qualifiers, 3-65
.TYPE entry syntax, 3-60, 3-63 to 3-65
differences between select and search, 3-61
signaling errors in, 3-65
.TYPE SEARCH, 3-6
SEARCH key
Gold S, 4-57
SEARCH132 arg form
and record collection output, 3-67

Searching text files
with search form .SFILE directive, 3-65
SEARCHKEY arg form
and record collection output, 3-67
Section
globally shared, 16-11
Segmented form
See Form set
SEL_CHOICE
FOR subfunction
and OA$SEL_LINE special symbol, 3-74
in select forms, 3-72
special FOR subfunction, 6-87
SEL_DISPLAY
FOR subfunction
and /SHOW, 4-45
in select forms, 3-72
special FOR subfunction, 6-87
SEL_STYLE
FOR subfunction
and /LIST, 3-70
in select forms, 3-72
special FOR subfunction, 6-87
Select form, 3-68 to 3-74
Named Data, 3-69
processing, 3-72
summary, 3-74
.TYPE line, 3-71
qualifiers, 3-69
/LIST, 3-69
/STYLE, 3-70
required features, 3-69
.TYPE SELECT, 3-6
Selection List
and NEXT_LIST, 6-155 to 6-156
created by /LIST select-form qualifier, 3-70
created by search form, 3-60
identifying
with /LIST field qualifier, 4-22
identifying next selection in, 4-23
opening and reading
with OA$FLD_NEXT_LIST, 4-60
Sequential files
See Text files
SET_DEFAULT function, 6-298 to 6-299
and COMMAND function, 6-36
and DCL function, 6-53
and NEWDIR, 6-150
and OAUSER, 6-299
SET_MENU function, 3-32, 6-300 to 6-301
.SFILE directive
in search form Named Data, 3-65
syntax, 3-66
Shared Document Attributes File
See SDAF
Shared libraries
in the File Cabinet, 9-19 to 9-20
/SHOW field qualifier, 4-45 to 4-46, 4-55
and SEL_DISPLAY, 4-45
with multiple data sets, 4-45
SHOW_FORM function, 3-4, 3-12, 6-302
and nonstandard forms, 3-80
SORT_LIST function, 6-303 to 6-305
See also MERGE_LIST, TEST_SPEC
.SOUT directive
in search form Named Data, 3-67
syntax, 3-67
SPAWN function, 6-306 to 6-307
and the primary subprocess, 7-19
Special symbol
table, 41

Special symbols, 8-3, 8-16 to 8-30
DTR$, 6-66
performance versus DSRs, 16-23
read-only, 8-15 to 8-30
read-write, 8-15 to 8-30
STAT$, 6-308
used with FOR function, 3-73
write-only, 8-15 to 8-30
Special-function keys
See Keyboard directives
Stacking functions, 6-4 to 6-9
See also Function lists
Stacking menu options, 3-32
restrictions when, 3-33
Statistics
ALL-IN-1 release
and VERSION, 6-316
data set
and OA$DSA_CACHE_
STATUS, 6-169 to 6-170
form library
and OA$FBT_LIST_
LIBTREE, 6-175 to 6-176
and OA$FBT_LIST_TREE,
6-177 to 6-178
process
and STATISTICS, 6-308 to
6-309
STATISTICS function, 6-308 to
6-309
/STORE field qualifier, 4-46
/STYLE select form qualifier
and /LIST, 3-71
and SEL_STYLE
subfunction, 3-72
used to format record
collection, 3-70
Subprocess, 7-1 to 7-23
accessing with DCL function,
6-51 to 6-53
calling function lists from, 6-8
creating an alternate
with SPAWN, 6-306
initializing, 7-1
obtaining user input while
in, 7-20 to 7-23
passing data to programs in,
7-20
receiving data from
through OAMAILBOX, 7-4
running command procedures
in, 7-19
sending data to
through DCLMAILBOX, 7-4
sending text files to
with INCLUDE, 6-115
trapping errors while in
with ERROR, 6-6, 6-76 to
6-77
Subsystem
defined, 1-6
Subsystems, 3-24
interaction of, 1-6
/SUPER form qualifier, 3-23
Support, terminal, B-2
Symbol table
local, 8-4
permanent, 8-2
opened during
initialization, 2-3
Symbols, 8-1 to 8-36
basic types, 8-2 to 8-5
DSRs, 8-4
field name, 8-2
local, 8-4
permanent, 8-2
quoted string, 8-2
special, 8-3
concatenating in lists, 8-11
creating, 8-14
with /COPY field qualifier,
4-8
dates and times in, 8-13
See also Date/time formats
defining and using
in the subprocess, 7-8 to 7-16

Symbols (Cont.)
getting values from, 8-15
with /GET_SAVE, 4-16
with GET, 6-101 to 6-107
in function calls, 6-12
in the main process, 8-15
in the subprocess, 8-15
iterative substitution, 8-12
performance differences, 8-15
putting values into, 8-14
with /PUT_SAVE, 4-27
substrings, 8-10
values stored in multiple, 8-15
zero-suppressing numeric, 8-11
Symbols, DCL
creating
with GET, 6-103
SYS$INPUT
mailbox to subprocess, 7-4

T

TAB
and OA$FLD_NEXT, 4-62
Terminal Modes
See Modes
Terminal Support, B-2, B-5
TERMINATOR
common DCL symbol, 7-13
and interactive functions, 7-13
DCL symbol
and PROMPT, 6-267, 6-271
set by OAFLD_DONE, OAFLD_EXIT, 4-67
Terminator functions
See OA$FLD_nnn
internal, 4-66
Terminator key(s)
processing, 4-66 to 4-69
in menu choice field, 3-31
TEST_SPEC function, 6-310 to 6-311
See also MERGE_LIST, SORT_LIST
Text data sets, 10-2
categories
ASCII, 10-2
COMPOUND, 10-2
DX, 10-2
DXT, 10-2
MAIL_ATTACHMENT, 10-2
MAIL_HEAD, 10-2
PIPELINE, 10-3
WPL, 10-3
OA$EDIT_BUFFER, 6-75
writing to
with DESTINATION, 6-58
Text files
copying
with COPY, 6-43 to 6-44
creating
with EDIT, 6-71 to 6-73, 10-3
with MERGE, 6-134 to 6-140
with MERGE_LINE, 6-141 to 6-142
with MERGE_LIST, 6-143 to 6-146
deleting, 6-56 to 6-57
displaying
See also OA$CNV_FROM_SCROLL
See also OA$CNV_TO_SCROLL
with LIST, 6-120
extensions and editors, 10-3
formatting, 10-28 to 10-33
including text into
with INCLUDE, 6-114 to 6-115
internal structure versus editor style, 10-21
modifying
See also OA$CNV_FROM_SCROLL
See also OA$CNV_TO_SCROLL
with EDIT, 6-71 to 6-73
(default editor), 10-4 to 10-19

Text files (Cont.)
naming
with /DOC, 4-11
printing
with QUEUE__PRINT
on an attached printer, 6-264
processing, 10-1 to 10-33
searching
with search form .SFILE
directive, 3-65
validating existence of
with /FILE__CHECK, 4-14
WPS-PLUS
and MERGE__LIST, 6-143
to 6-146
formatting
with FORMAT, 6-97 to
6-100
Text processing, 10-1 to 10-33
editor menu, 6-74 to 6-75
Time
formatting
See Dates and times
TIME function, 6-312 to 6-313
Time Management, 13-1 to 13-30
activities, 13-2
CALENDAR, 6-26
organization, 13-1
/TITLE form qualifier, 3-23
TRACE keys, VERIFY and
Gold (,Gold), 4-62
TXL
and the OAGBL.BLI
module, 15-5
building and modifying, 15-5
search order, 15-6
changing with
OA$TXL__SEARCH__LAST,
15-6
to increase performance, 15-5
.TYPE or type
See Form .TYPE

U

/UNIQUE field qualifier, 4-47, 4-52
UNWIND function, 6-314 to 6-315
Up Arrow
and OA$FLD__UP, 4-62
calls OA$SCL__UP in scrolled
region, 4-34
/USE__FORM field qualifier, 4-47
/USE__FORM validation
subqualifier, 4-21, 4-49, 4-51, 4-55
with /INVALID, 4-19
with /RECOGNITION, 4-28
with /RSE__nnn, 4-31
/USER form qualifier, 3-24
User Profile
ACTITEM field, 6-18
and NEWLIB, 6-152 to 6-154
DOCFRM field
and NEWDIR, 6-150
Form Libraries field (FRMLIB)
use of, 3-3
FRMLIB field, 16-9
opened during initialization, 2-2
privileges
PRVCMD, 3-23, 3-32, 6-2
PRVDCL, 3-32, 4-58
PRVSUP, 3-23
User-defined functions
See Functions, user-defined
User-defined keys
See Keyboard directives

V

/VALID field qualifier, 4-4, 4-47
to 4-50
and /RECOGNITION, 4-49
on entry form key fields, 3-45
with /RECOGNITION, 11-50
Validation
and OA$VAL__SET__VALID,
6-260
field qualifiers, 4-4
/AUTO, 4-5
combining, 4-54
/FILE__CHECK, 4-14 to 4-16
/INVALID, 4-19 to 4-22
/OPTIONAL, 4-23 to 4-24

Validation
field qualifiers (Cont.)
/RECOGNITION, 4-28 to 4-30
/RSE_INVALID, 4-30 to 4-32
/RSE_RECOG, 4-30 to 4-32
/RSE_VALID, 4-30 to 4-32
/UNIQUE, 4-47
/VALID, 4-47 to 4-50
optional
with /OPTIONAL, 4-23
with /RECOGNITION, 4-28
through field qualifiers, 4-51 to 4-56
with RSEs, 4-30 to 4-32
Validation expression, 4-51, 45
DSR in a, 4-20, 4-29, 4-48, 4-51
with /INVALID, 4-19
with /RECOGNITION, 4-28
with /VALID, 4-47
VAX Information Architecture
and ALL-IN-1, 1-4
overview, 1-3
VERIFY and TRACE keys
Gold (,Gold), 4-62
Verifying
command procedures
with Gold (, 6-34
Version 2.X, converting to
form libraries, 16-9
VERSION function, 6-316 to 6-317
View form, 16-18
See also individual forms
.TYPE VIEW, 3-6
VMS
file names
See File names

W

WAIT function, 6-270, 6-318 to 6-320
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13
WPINDX
select form
processing of, 3-72
WPS-PLUS
and MERGE_LIST, 6-143 to 6-146
and SORT_LIST, 6-303 to 6-305
and TEST_SPEC, 6-310 to 6-311
WPSPLUS$CBI_CLOSE_WINDOW, 6-321
WPSPLUS$CBI_CLOSE_WINDOW2
function, 6-322
WPSPLUS$CBI_OPEN_WINDOW
function, 6-323 to 6-324
WPSPLUS$CBI_OPEN_WINDOW2
function, 6-325 to 6-326
WPSPLUS_SPELL function, 6-327
WRITE function, 6-328 to 6-329
and entry-form modes, 3-78
called by FOR, 6-91

Y

YESNO_PROMPT function, 6-270, 6-330 to 6-332
See also PROMPT
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13
YESNO_PROMPT_ID
function, 6-333 to 6-334
See also PROMPT
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13

Text files (Cont.)
naming
with /DOC, 4-11
printing
with QUEUE_PRINT
on an attached printer, 6-264
processing, 10-1 to 10-33
searching
with search form .SFILE directive, 3-65
validating existence of
with /FILE_CHECK, 4-14
WPS-PLUS
and MERGE_LIST, 6-143 to 6-146
formatting
with FORMAT, 6-97 to 6-100
Text processing, 10-1 to 10-33
editor menu, 6-74 to 6-75
Time
formatting
See Dates and times
TIME function, 6-312 to 6-313
Time Management, 13-1 to 13-30
activities, 13-2
CALENDAR, 6-26
organization, 13-1
/TITLE form qualifier, 3-23
TRACE keys, VERIFY and Gold (,Gold), 4-62
TXL
and the OAGBL.BLI module, 15-5
building and modifying, 15-5
search order, 15-6
changing with OA$TXL_SEARCH_LAST, 15-6
to increase performance, 15-5
.TYPE or type
See Form .TYPE

U

/UNIQUE field qualifier, 4-47, 4-52
UNWIND function, 6-314 to 6-315
Up Arrow
and OA$FLD_UP, 4-62
calls OA$SCL_UP in scrolled region, 4-34
/USE_FORM field qualifier, 4-47
/USE_FORM validation subqualifier, 4-21, 4-49, 4-51, 4-55
with /INVALID, 4-19
with /RECOGNITION, 4-28
with /RSE_nnn, 4-31
/USER form qualifier, 3-24
User Profile
ACTITEM field, 6-18
and NEWLIB, 6-152 to 6-154
DOCFRM field
and NEWDIR, 6-150
Form Libraries field (FRMLIB)
use of, 3-3
FRMLIB field, 16-9
opened during initialization, 2-2
privileges
PRVCMD, 3-23, 3-32, 6-2
PRVDCL, 3-32, 4-58
PRVSUP, 3-23
User-defined functions
See Functions, user-defined
User-defined keys
See Keyboard directives

V

/VALID field qualifier, 4-4, 4-47 to 4-50
and /RECOGNITION, 4-49
on entry form key fields, 3-45
with /RECOGNITION, 11-50

Validation
and OA$VAL_SET_VALID, 6-260
field qualifiers, 4-4
/AUTO, 4-5
combining, 4-54
/FILE_CHECK, 4-14 to 4-16
/INVALID, 4-19 to 4-22
/OPTIONAL, 4-23 to 4-24
/RECOGNITION, 4-28 to 4-30
/RSE_INVALID, 4-30 to 4-32
/RSE_RECOG, 4-30 to 4-32
/RSE_VALID, 4-30 to 4-32
/UNIQUE, 4-47
/VALID, 4-47 to 4-50
optional
with /OPTIONAL, 4-23
with /RECOGNITION, 4-28
through field qualifiers, 4-51 to 4-56
with RSEs, 4-30 to 4-32
Validation expression, 4-51, 45
DSR in a, 4-20, 4-29, 4-48, 4-51
with /INVALID, 4-19
with /RECOGNITION, 4-28
with /VALID, 4-47
VAX Information Architecture
and ALL-IN-1, 1-4
overview, 1-3
VERIFY and TRACE keys
Gold (,Gold), 4-62
Verifying
command procedures
with Gold (, 6-34
Version 2.X, converting to
form libraries, 16-9
VERSION function, 6-316 to 6-317
View form, 16-18
See also individual forms
.TYPE VIEW, 3-6
VMS
file names
See File names

W

WAIT function, 6-270, 6-318 to 6-320
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13
WPINDX
select form
processing of, 3-72
WPS-PLUS
and MERGE_LIST, 6-143 to 6-146
and SORT_LIST, 6-303 to 6-305
and TEST_SPEC, 6-310 to 6-311
WPSPLUS$CBI_CLOSE_WINDOW, 6-321
WPSPLUS$CBI_CLOSE_WINDOW2 function, 6-322
WPSPLUS$CBI_OPEN_WINDOW function, 6-323 to 6-324
WPSPLUS$CBI_OPEN_WINDOW2 function, 6-325 to 6-326
WPSPLUS$SPELL functions, 6-327
WRITE function, 6-328 to 6-329
and entry-form modes, 3-78
called by FOR, 6-91

Y

YESNO_PROMPT function, 6-270, 6-330 to 6-332
See also PROMPT
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13
YESNO_PROMPT_ID function, 6-333 to 6-334
See also PROMPT
and OA$FLD_nnn functions, 4-67
from the subprocess, 7-13

HOW TO ORDER ADDITIONAL DOCUMENTATION

DIRECT TELEPHONE ORDERS

In Continental USA and Puerto Rico call **800–258–1710**

In Canada call **800–267–6146**

In New Hampshire, Alaska or Hawaii call **603–884–6660**

DIRECT MAIL ORDERS (U.S. and Puerto Rico*)

DIGITAL EQUIPMENT CORPORATION
P.O. Box CS2008
Nashua, New Hampshire 03061

DIRECT MAIL ORDERS (Canada)

DIGITAL EQUIPMENT OF CANADA LTD.
940 Belfast Road
Ottawa, Ontario, Canada K1G 4C2
Attn: P&SG Business Manager

INTERNATIONAL

DIGITAL EQUIPMENT CORPORATION
P&SG Business Manager
c/o Digital's local subsidiary
or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

*Any prepaid order from Puerto Rico must be placed with the Local Digital Subsidiary:
809–754–7575

Reader's Comments

Your comments and suggestions are welcome. They will help us in our continuous effort to improve the quality and usefulness of our publications.

1 How would you rate this manual for:

	good	fair	poor
completeness of information	_____	_____	_____
accuracy of information	_____	_____	_____
ease of use (clarity, organization)	_____	_____	_____

2 Did you find errors in this manual? Please specify by page and paragraph, or mark corrections on pages and attach copies to this form.

	page	paragraph
a) Incorrect information:	_____	_____
b) Information left out:	_____	_____
c) Hard to understand:	_____	_____

3 What suggestions do you have for improving this manual?

_____

_____

4 What parts of the manual were especially good at helping you understand things?

_____

_____

5 Please check the boxes that apply to you.

- ☐ Technical
- ☐ Nontechnical
- ☐ Management
- ☐ Nonmanagement
- ☐ Other (explain)_____
- ☐ Used other word processors
- ☐ Used computers
- ☐ Use the word processing unit yourself

Name_____ Title_____

Company_____

Street_____ City_____

State_____ Zip_____ Date_____

Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Manager OFFICE PROGRAM ZK02-1/N20
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, N.H. 03062

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line

AD-DC94A-T1

OFFICE MENU

Application Programmer's Reference Volume 3: Applications

Update Package #1, January 1986

This package updates the ALL-IN-1 Office Menu Application Programmer's Reference Volume 3 (Order No AA-DC94A-TE) for ALL-IN-1 Version 2.1.

In this package, a bar (▌) in the outside margin shows where information has been changed or added. A bullet (●) shows where information has been deleted. These marks do not appear in the Table of Contents or Index.

Use the pages in this package to update the Getting Started Guide. Discard the pages that you have replaced.

Insert the pages as follows.

Insert page	In place of
Guide to ALL-IN-1 Information	Guide to ALL-IN-1 Information
iii – xlii	iii – xlii
xlvii – l	xlvii – l
8-15 – 8-16	8-15 – 8-16
8-23 – 8-24	8-23 – 8-24

(continued)

Insert page	In place of
9-19 – 9-20	9-19 – 9-20
11-11 – 11-16.1	11-11 – 11-16.1
11-47 – 11-51	11-47 – 11-50
13-17 – 13-20.1	13-17 – 13-20
15-5 – 15-6	15-5 – 15-6
15-13 – 15-14.1	15-13 – 15-14
B-3 – B-4	B-3 – B-4
Glossary-17 – 20	Glossary-17 – 20
Index-1 – 36	Index-1 – 38

AD-DC94A-T2

ALL-IN-1

OFFICE MENU

Application Programmer's Reference
Volume 3: Applications

Update Package #2, April 1987

Update the Application Programmer's Reference Volume 2 (AA-DC93A-TE) with this package. You must first have updated the APR with update package #1, January 1986 (AD-DC94A-T1). When you have inserted both packages, the APR will be updated for ALL-IN-1 Version 2.2.

In this package, a bar (▮) in the outside margin shows where information has been changed or added. A bullet (•) shows where information has been deleted. These marks do not appear in the Table of Contents or Index.

Insert the pages as follows.

Insert page	In place of
xxi – xxii	xxi – xxii
7-3 – 7-4	7-3 - 7-4
8-17 – 8-20	8-17 – 8-20
9-13 – 9-14	9-13 – 9-14
9-17 – 9-18	9-17 – 9-18
9-21 – 9-22	9-21 – 9-22
9-43 – 9-47	9-43 – 9-47
10-25 – 10-26	10-25 – 10-26
13-1 – 13-4	13-1 – 13-4

(continued)

Insert page	In place of
13-20.1	13-20.1
13-23 – 13-24	13-23 – 13-24
13-27 – 13-30	13-27 – 13-30
14-23 – 14-26	14-23 – 14-26
15-5 – 15-6	15-5 – 15-6
15-23 – 15-24	15-23 – 15-24
16-9 – 16-12	16-9 – 16-12
16-19 – 16-20.1	16-19 – 16-20
Glossary-35 – 36	Glossary-35 – 36
Glossary-41 – 42	Glossary-41 – 42
Index-15 – 16	Index-15 – 16
Index-21 – 22	Index-21 – 22
Index-25 – 28	Index-25 – 28

AA-DC94A-TE
Printed in U.S.A.